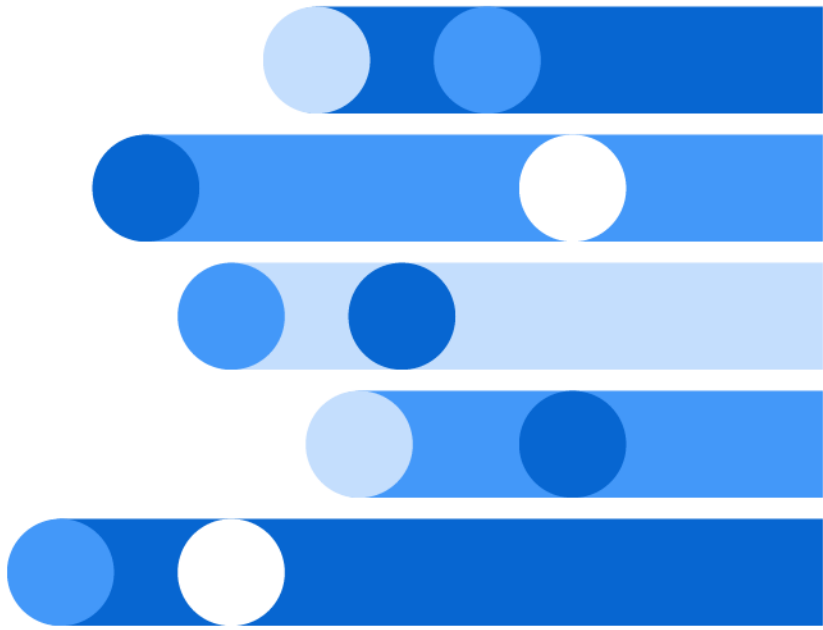
SAS® Viya® Platform: Network Python API Guide

2026.06*



* This document might apply to additional versions of the software. Open this document in SAS Help Center and click on the version in the banner to see all available versions.

SAS® Documentation
July 28, 2026

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2026. *SAS® Viya® Platform: Network Python API Guide*. Cary, NC: SAS Institute Inc.

SAS® Viya® Platform: Network Python API Guide

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

July 2026

v_008-P1:ecepynetpg

Contents

1

Biconnected Component Algorithms

Functions

Function Name	Description
articulation_points	Finds the articulation points (cut points) of a graph.
biconcomp_distribution_edges	Computes the distribution of the number of edges in each biconnected components by using *numpy.histogram*.
biconcomp_distribution_nodes	Computes the distribution of the number of nodes in each biconnected components by using *numpy.histogram*.
biconcomp_size_edges	Computes the number of edges within each biconnected component.
biconcomp_size_nodes	Computes the number of nodes within each biconnected component.
biconnected_components	Computes the graph's biconnected components for an undirected graph.
block_cut_tree	Computes the block-cut tree of a graph.
get_nlargest_biconcomps	Returns a list of up to *n* largest biconnected components.
is_biconnected	Specifies whether or not the input graph is biconnected.

articulation_points

Finds the articulation points (cut points) of a graph.

API: articulation_points

```
sasviya.network.algorithms.biconnected_components.articulation_points(g)
```

Parameters

g : SAS graph object
Specifies the undirected input graph to assess.

Returns

list
Returns a list of articulation points in the graph.

biconcomp_distribution_edges

Computes the distribution of the number of edges in each biconnected components by using *numpy.histogram*.

API: biconcomp_distribution_edges

sasviya.network.algorithms.biconnected_components.biconcomp_distribution_edges(g, bins=None)

Parameters

g : SAS graph object
Specifies the undirected input graph to assess.

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is int (an integer), it specifies the number of equal-width bins. When *bins* is a list, it specifies bin ranges and should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is str (a string), it specifies the method to use for calculating the bin ranges. For example, 'stone','auto','doane','fd','rice','scott','sqrt','sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of biconnected components whose edge count is within the range of the bin.

biconcomp_distribution_nodes

Computes the distribution of the number of nodes in each biconnected components by using *numpy.histogram*.

API: biconcomp_distribution_nodes

sasviya.network.algorithms.biconnected_components.biconcomp_distribution_nodes(g, bins=None)

Parameters

g : SAS graph object
Specifies the undirected input graph to assess.

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is int (an integer), it specifies the number of equal-width bins. When *bins* is a list, it specifies bin ranges and should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is str (a string), it specifies the method to use for calculating the bin ranges. For example, 'stone','auto','doane','fd','rice','scott','sqrt','sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of biconnected components whose node count is within the range of the bin.

biconcomp_size_edges

Computes the number of edges within each biconnected component.

API: biconcomp_size_edges

```
sasviya.network.algorithms.biconnected_components.biconcomp_size_edges(g)
```

Parameters

g : SAS graph object
Specifies the undirected input graph to assess.

Returns

dict
Returns a dictionary in which the keys are biconnected component IDs and the values are the number of edges within each component.

biconcomp_size_nodes

Computes the number of nodes within each biconnected component.

API: biconcomp_size_nodes

```
sasviya.network.algorithms.biconnected_components.biconcomp_size_nodes(g)
```

Note that articulation points occur within more than one biconnected component.

Parameters

g : SAS graph object
Specifies the undirected input graph to assess.

Returns

dict
Returns a dictionary in which the keys are biconnected component IDs and the values are the number of nodes within each component.

biconnected_components

Computes the graph's biconnected components for an undirected graph.

API: biconnected_components

```
sasviya.network.algorithms.biconnected_components.biconnected_components(g, order_by=None, *, data=True, ascending=False)
```

Parameters

g : SAS graph object
Specifies the undirected input graph to assess.

order_by : 'node' or 'edge' or None, optional
Specifies the parameter that is used to sort the components by the number of nodes or edges within each component. The default is None, which means that the components are not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Returns the result as a generator of biconnected component subgraphs. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

block_cut_tree

Computes the block-cut tree of a graph.

API: block_cut_tree

```
sasviya.network.algorithms.biconnected_components.block_cut_tree(g, *,
return_nodes_map=False)
```

Parameters

g : SAS graph object
Specifies the undirected input graph to assess.

return_nodes_map : bool, optional
Specifies whether or not to also compute the block-cut tree node map. The default is False.

Returns

SAS graph object
Returns a graph that represents the block-cut tree. When *return_nodes_map = True*, the output graph has the *nodes_map* attribute, which contains a dictionary that maps its nodes to a list of nodes in the original graph. Note: If a block is empty (that is, the corresponding biconnected component contains only articulation points), that block does not appear as a key in the *nodes_map* dictionary.

get_nlargest_biconcomps

Returns a list of up to *n* largest biconnected components.

API: get_nlargest_biconcomps

```
sasviya.network.algorithms.biconnected_components.get_nlargest_biconcomps(g, n, order_by='node', *, data=True, ascending=False)
```

These are the components that have the highest number of nodes or edges of a graph. When ties occur, the component that has the lower *biconcomp_id* value is returned.

Parameters

g : SAS graph object
Specifies the input graph.

n : int
Specifies the number of graphs to return.

order_by : 'node' or 'edge' or None
Specifies whether to sort the biconnected components by the number of nodes or edges within those components. The default is None, which means that the biconnected components are not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. When the value is True, the smallest biconnected components are returned instead. The default is False.

Returns

list
Returns a list of n largest or smallest biconnected components of a graph.

is_biconnected

Specifies whether or not the input graph is biconnected.

API: is_biconnected

```
sasviya.network.algorithms.biconnected_components.is_biconnected(g)
```

A biconnected graph is one that cannot be disconnected by removing only one node (plus all incident edges).

Parameters

g : SAS graph object
Specifies the input undirected graph to assess.

Returns

bool
Indicates whether the input graph is biconnected (True) or not (False).

2

Centrality Algorithms

Functions

Function Name	Description
centrality_betweenness	Computes weighted or unweighted betweenness centrality and stores the results as attributes on the graph.
centrality_closeness	Computes weighted or unweighted closeness centrality and stores the results as attributes on the graph.
centrality_degree	Computes weighted or unweighted degree centrality and stores the results as attributes on the graph.
centrality_eigenvector	Computes weighted or unweighted eigenvector centrality and stores the results as attributes on the graph.
centrality_hub_authority	Computes weighted or unweighted hub and authority centrality and stores the results as attributes on the graph.
centrality_influence	Computes weighted or unweighted influence centrality and stores the results as attributes on the graph.
centrality_pagerank	Computes weighted or unweighted PageRank centrality and stores the results as attributes on the graph.

centrality_betweenness

Computes weighted or unweighted betweenness centrality and stores the results as attributes on the graph.

API: centrality_betweenness

```
sasviya.network.algorithms.centralities.centrality_betweenness(g,
edge_weight=None, sample_percent=100, *, inplace=False, data=True)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that unweighted betweenness centrality is computed.

sample_percent : int, optional
Specifies the percentage of source nodes to sample for the approximate betweenness calculation. This parameter value should be in the range (0, 100]. The default is 100.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the node and edge attributes *between_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_closeness

Computes weighted or unweighted closeness centrality and stores the results as attributes on the graph.

API: centrality_closeness

```
sasviya.network.algorithms.centralities.centrality_closeness(g,
edge_weight=None, nopath='diameter', *, inplace=False, data=True)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted closeness centrality is computed.

nopath : {'diameter', 'harmonic', or 'zero'}, optional
Specifies a method to use for accounting for the shortest path distance between two nodes when a path does not exist. The valid values are as follows: - 'diameter' uses the graph diameter (plus one) as the shortest path distance between disconnected nodes. - 'harmonic' uses the harmonic formula for calculating closeness centrality. - 'zero' uses zero as the shortest path distance between disconnected nodes.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the closeness centrality node attributes *centr_close_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_degree

Computes weighted or unweighted degree centrality and stores the results as attributes on the graph.

API: centrality_degree

```
sasviya.network.algorithms.centralities.centrality_degree(g, edge_weight=None,
*, inplace=False, data=True)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted degree centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes

of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the degree centrality node attributes *centr_degree_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_eigenvector

Computes weighted or unweighted eigenvector centrality and stores the results as attributes on the graph.

API: centrality_eigenvector

```
sasviya.network.algorithms.centralities.centrality_eigenvector(g,
edge_weight=None, maxiters=10000, *, inplace=False, data=True)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted eigenvector centrality is computed.

maxiters : int, optional
Specifies the maximum number of iterations in order to limit the amount of computation time that is spent when convergence is slow. The default is 10,000.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the degree centrality node attributes *centr_eigen_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_hub_authority

Computes weighted or unweighted hub and authority centrality and stores the results as attributes on the graph.

API: centrality_hub_authority

```
sasviya.network.algorithms.centralities.centrality_hub_authority(g,
edge_weight=None, *, inplace=False, data=True)
```

Parameters

g : SAS graph object
Specifies the directed input graph to assess.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted hub and authority centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes

of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the hub and authority centrality node attributes *centr_hub_** and *centr_auth_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_influence

Computes weighted or unweighted influence centrality and stores the results as attributes on the graph.

API: centrality_influence

```
sasviya.network.algorithms.centralities.centrality_influence(g,
edge_weight=None, *, inplace=False, data=True)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted influence centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the first- and second-order influence centrality node attributes *centr_influence1_** and *centr_influence2_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_pagerank

Computes weighted or unweighted PageRank centrality and stores the results as attributes on the graph.

API: centrality_pagerank

```
sasviya.network.algorithms.centralities.centrality_pagerank(g,
edge_weight=None, alpha=0.85, tolerance=1e-09, *, inplace=False, data=True)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted PageRank centrality is computed.

alpha : float, optional
Specifies the damping parameter for the PageRank centrality (or algorithm). The default is 0.85.

tolerance : float, optional
Specifies the tolerance parameter for the PageRank centrality (or algorithm). The default is 1E-9.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional

Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None

Returns a graph that contains the PageRank centrality node attributes *centr_pagerank_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

3

Clique Algorithms

Functions

Function Name	Description
maximal_cliques	Enumerates all maximal cliques of the graph.
node_clique_numbers	Computes node clique numbers and stores the results as attributes on the graph.

maximal_cliques

Enumerates all maximal cliques of the graph.

API: maximal_cliques

```
sasviya.network.algorithms.cliques.maximal_cliques(g, maxcliques='ALL',
node_weight=None, edge_weight=None, minsize=1, maxsize=None,
minedgewt=None, maxedgewt=None, minnodewt=None, maxnodewt=None,
maxtime=None, order_by=None, *, data=True, ascending=False)
```

Parameters

g : SAS graph object
Specifies the simple and undirected input graph to assess.

maxcliques : int or 'ALL', optional
Specifies the maximum number of enumerated cliques to return. The default is 'ALL'.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

minsize : int, optional
Specifies the minimum number of nodes in a clique. The default is 1.

maxsize : int or None, optional
Specifies the maximum number of nodes in a clique. The default is None.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a clique. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a clique. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a clique. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a clique. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds for enumeration to spend. The default is None.

order_by : 'node' or None, optional
Specifies whether to sort the cliques by the number of nodes within each clique. The default is None, which means that the result is not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one maximal clique. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

node_clique_numbers

Computes node clique numbers and stores the results as attributes on the graph.

API: node_clique_numbers

```
sasviya.network.algorithms.cliques.node_clique_numbers(g, *, inplace=False, data=True)
```

Parameters

g : SAS graph object
Specifies the simple and undirected input graph to assess.

inplace : bool, optional
Specifies whether to add the clique number values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the value of the *inplace* parameter is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None

Returns a graph or modifies the existing graph to contain the node attribute *clique_number*, which includes the clique number of each node. The graph g has the attribute g.solution_summary, which contains information about the results of the algorithm.

4

Community Algorithms

Functions

Function Name	Description
label_propagation	Computes the communities of the graph by using the label propagation algorithm.
label_propagation_nodes	Computes community assignments for all nodes in the graph by using the label propagation algorithm.
louvain	Computes the communities of the graph by using the Louvain algorithm.
louvain_nodes	Computes community assignments for all nodes in the graph by using the Louvain algorithm.

label_propagation

Computes the communities of the graph by using the label propagation algorithm.

API: label_propagation

```
sasviya.network.algorithms.communities.label_propagation(g,
edge_weight=None, maxiters=100, tolerance=0.05, *, data=True)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of label changes for all nodes in the graph is less than the value. The default is 0.05.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one community. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

label_propagation_nodes

Computes community assignments for all nodes in the graph by using the label propagation algorithm.

API: label_propagation_nodes

```
sasviya.network.algorithms.communities.label_propagation_nodes(g,
edge_weight=None, maxiters=100, tolerance=0.05, *, data=True, inplace=False)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of label changes for all nodes in the graph is less than the value. The default is 0.05.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

inplace : bool, optional
Specifies whether to add the community values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains the node attribute *community*, which denotes the community assignment for each node. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

louvain

Computes the communities of the graph by using the Louvain algorithm.

API: louvain

```
sasviya.network.algorithms.communities.louvain(g, *, edge_weight=None,
resolution=1, tolerance=0.001, maxiters=100, data=True)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

resolution : float, optional
Specifies the factor in the modularity equation that affects the expected number of links between two nodes. A higher value produces more communities. The default is 1.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of modularity gain between two consecutive iterations is less than the value. The default is 0.001.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one community. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

louvain_nodes

Computes community assignments for all nodes in the graph by using the Louvain algorithm.

API: louvain_nodes

```
sasviya.network.algorithms.communities.louvain_nodes(g, *,
edge_weight=None, resolution=1, tolerance=0.001, maxiters=100, data=True,
inplace=False)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

resolution : float, optional
Specifies the factor in the modularity equation that affects the expected number of links between two nodes. A higher value produces more communities. The default is 1.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of modularity gain between two consecutive iterations is less than the value. The default is 0.001.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

inplace : bool, optional
Specifies whether to add the community values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains the node attribute *community*, which denotes the community assignment for each node. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

5

Connected Component Algorithms

Functions

Function Name	Description
concomp_distribution_edges	Computes the distribution of the number of edges in each connected component by using *numpy.histogram*.
concomp_distribution_nodes	Computes the distribution of the number of nodes in each connected component by using *numpy.histogram*.
concomp_size_edges	Computes the number of edges within each connected component.
concomp_size_nodes	Computes the number of nodes within each connected component.
connected_components	Computes the connected components of a graph.
get_nlargest_concomps	Returns a list of up to *n* largest connected (with highest number of nodes) components of a graph.
is_connected	Specifies whether or not the input graph is connected.
is_strongly_connected	Specifies whether the input graph is strongly connected.
is_weakly_connected	Specifies whether the input graph is weakly connected.

concomp_distribution_edges

Computes the distribution of the number of edges in each connected component by using *numpy.histogram*.

API: concomp_distribution_edges

```
sasviya.network.algorithms.connected_components.concomp_distribution_edges(g, bins=None, *, strongly=True)
```

For directed graphs, strongly or weakly connected components can be computed.

Parameters

g : SAS graph object
Specifies the input graph to assess.

bins : int or list or str, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is an integer (int), it represents the number of equal-width bins. When *bins* is a list, it specifies bin ranges and it should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is a string (str), it specifies the method to be used for calculating the bin ranges. For example, 'stone', 'auto', 'doane', 'fd', 'rice', 'scott', 'sqrt', 'sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of components whose edge count within the range of the bin.

concomp_distribution_nodes

Computes the distribution of the number of nodes in each connected component by using *numpy.histogram*.

API: concomp_distribution_nodes

```
sasviya.network.algorithms.connected_components.concomp_distribution_nodes(g, bins=None, *, strongly=True)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is an integer (int), it represents the number of equal-width bins. When *bins* is a list, it specifies bin ranges and it should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is a string (str), it specifies the method to use for calculating the bin ranges. For example, 'stone', 'auto', 'doane', 'fd', 'rice', 'scott', 'sqrt', 'sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of components whose node count is within the range of the bin.

concomp_size_edges

Computes the number of edges within each connected component.

API: concomp_size_edges

```
sasviya.network.algorithms.connected_components.concomp_size_edges(g, *,
strongly=True)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary in which the keys are connected component IDs and the values are the number of edges within each component.

concomp_size_nodes

Computes the number of nodes within each connected component.

API: concomp_size_nodes

```
sasviya.network.algorithms.connected_components.concomp_size_nodes(g, *,
strongly=True)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary in which the keys are connected component IDs and the values are the number of nodes within each component.

connected_components

Computes the connected components of a graph.

API: connected_components

```
sasviya.network.algorithms.connected_components.connected_components(g,
order_by=None, *, strongly=True, data=True, ascending=False)
```

For directed graphs, either strongly or weakly connected components can be computed.

Parameters

g : SAS graph object
Specifies the input graph to assess.

order_by : 'node' or 'edge' or None, optional
Specifies whether to sort the components by the number of nodes or edges within each component. The default is None, which means that the components are not sorted.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one connected component. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

get_nlargest_concomps

Returns a list of up to *n* largest connected (with highest number of nodes) components of a graph.

API: get_nlargest_concomps

```
sasviya.network.algorithms.connected_components.get_nlargest_concomps(g,
n, order_by='node', *, strongly=True, data=True, ascending=False)
```

When ties occur, the component with the lower concomp ID is returned.

Parameters

g : SAS graph object
Specifies the input graph to assess.

n : int
Specifies the number of graphs to return.

order_by : 'node' or 'edge'
Specifies whether to sort the component by number of nodes or edges within that component. The default is None, which means that the result is not sorted.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Returns

list
Returns a list of the n largest connected components of a graph.

is_connected

Specifies whether or not the input graph is connected.

API: is_connected

```
sasviya.network.algorithms.connected_components.is_connected(g)
```

Parameters

g : SAS graph object
Specifies the undirected input graph to assess.

Returns

bool
Indicates whether the input graph is connected (True) or not (False).

is_strongly_connected

Specifies whether the input graph is strongly connected.

API: is_strongly_connected

```
sasviya.network.algorithms.connected_components.is_strongly_connected(g)
```

Parameters

g : SAS graph object
Specifies the directed input graph to assess.

Returns

bool
Indicates whether the input graph is strongly connected (True) or not (False).

is_weakly_connected

Specifies whether the input graph is weakly connected.

API: is_weakly_connected

```
sasviya.network.algorithms.connected_components.is_weakly_connected(g)
```

Parameters

g : SAS graph object
Specifies the directed input graph to assess.

Returns

bool
Indicates whether the input graph is weakly connected (True) or not (False).

6

Core Algorithms

Functions

Function Name	Description
add_core_number_attr	Computes the core numbers and stores the results as node attributes on the graph.
core_decomposition	Computes a core decomposition of the graph.

add_core_number_attr

Computes the core numbers and stores the results as node attributes on the graph.

API: add_core_number_attr

```
sasviya.network.algorithms.cores.add_core_number_attr(g, maxtime=None, *,
inplace=False, data=True)
```

Parameters

g : SAS graph object
Specifies the input graph for which to calculate core numbers.

maxtime : float or None, optional
Specifies the maximum amount of time for core decomposition to spend. The default is None.

inplace : bool, optional
Specifies whether to add the core numbers as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph with core numbers that are stored as a node attribute named *core*. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

core_decomposition

Computes a core decomposition of the graph.

API: core_decomposition

```
sasviya.network.algorithms.cores.core_decomposition(g, maxtime=None, *,
data=True)
```

Parameters

g : SAS graph object
Specifies the input graph for which to calculate the core decomposition.

maxtime : float or None, optional
Specifies the maximum amount of time for core decomposition to spend. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one k-core in increasing core number. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

7

Cycle Algorithms

Functions

Function Name	Description
cycle_count	Counts the number of cycles in a graph.
enumerate_cycle_edges	Enumerates the cycles in a graph and returns the edges of the cycles.
enumerate_cycle_nodes	Enumerates the cycles in a graph and returns the nodes of the cycles.
enumerate_cycles	Enumerates the cycles in a graph and returns the cycle graph objects.

cycle_count

Counts the number of cycles in a graph.

API: cycle_count

```
sasviya.network.algorithms.cycles.cycle_count(g, maxcycles='ALL', minlength=1,
maxlength=None, node_weight=None, edge_weight=None, minedgewt=None,
maxedgewt=None, minnodewt=None, maxnodewt=None, maxtime=None,
source=None)
```

Parameters

g : SAS graph object
Specifies the input graph for which to count cycles.

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Returns

int
Returns a count of cycles in the graph.

enumerate_cycle_edges

Enumerates the cycles in a graph and returns the edges of the cycles.

API: enumerate_cycle_edges

```
sasviya.network.algorithms.cycles.enumerate_cycle_edges(g, maxcycles='ALL',
minlength=1, maxlength=None, node_weight=None, edge_weight=None,
minedgewt=None, maxedgewt=None, minnodewt=None, maxnodewt=None,
maxtime=None, source=None)
```

Parameters

g : SAS graph object
Specifies the input graph for which to count cycles.

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Yields

generator
Yields a generator of cycles, each of which is given as a list of the cycles' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_cycle_nodes

Enumerates the cycles in a graph and returns the nodes of the cycles.

API: enumerate_cycle_nodes

```
sasviya.network.algorithms.cycles.enumerate_cycle_nodes(g, maxcycles='ALL',
minlength=1, maxlength=None, node_weight=None, edge_weight=None,
minedgewt=None, maxedgewt=None, minnodewt=None, maxnodewt=None,
maxtime=None, source=None)
```

Parameters

g : SAS graph object
Specifies the input graph for which to count cycles.

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Yields

generator
Yields a generator of cycles, each of which is given as a list of the cycles' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_cycles

Enumerates the cycles in a graph and returns the cycle graph objects.

API: enumerate_cycles

```
sasviya.network.algorithms.cycles.enumerate_cycles(g, maxcycles='ALL',
minlength=1, maxlength=None, node_weight=None, edge_weight=None,
minedgewt=None, maxedgewt=None, minnodewt=None, maxnodewt=None,
maxtime=None, source=None, *, data=True)
```

Parameters

g : SAS graph object
Specifies the input graph for which to count cycles.

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one cycle of the same type as the input graph. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

8

Flow Algorithms

Functions

Function Name	Description
maxflow	Calculates the maximum flow from a source node to a sink node in a graph.
mincostflow	Calculates the minimum-cost network flow for the graph.

maxflow

Calculates the maximum flow from a source node to a sink node in a graph.

API: maxflow

```
sasviya.network.algorithms.flows.maxflow(g, source, sink, capacity_attr='upper',
*, inplace=False, data=True)
```

Parameters

g : SAS graph object
Specifies the input graph for which to calculate the maximum flow.

source : number or str
Specifies the source node for the maximum flow.

sink : number or str
Specifies the sink node for the maximum flow.

capacity_attr : str, optional
Specifies the name of the edge attribute that contains the capacity of the edge. The default is 'upper'.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute *flow*, which contains the flow values. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, the returned graph contains only the flow edges.

mincostflow

Calculates the minimum-cost network flow for the graph.

API: mincostflow

```
sasviya.network.algorithms.flows.mincostflow(g, demand_lower_attr='lower',
demand_upper_attr='upper', capacity_lower_attr='lower',
capacity_upper_attr='upper', cost_attr='weight', maxtime=None, *,
inplace=False, data=True)
```

Parameters

g : SAS graph object
Specifies the input directed graph for which to calculate the minimum-cost network flow.

demand_lower_attr : str, optional
Specifies the name of the node attribute that contains the lower bound of the node supply. The default is 'lower'.

demand_upper_attr : str, optional
Specifies the name of the node attribute that contains the upper bound of the node supply. The default is 'upper'.

capacity_lower_attr : str, optional
Specifies the name of the edge attribute that contains the capacity lower bound of the edge. The default is 'lower'.

capacity_upper_attr : str, optional
Specifies the name of the edge attribute that contains the capacity upper bound of the edge. The default is 'upper'.

cost_attr : str, optional
Specifies the name of the edge attribute that contains the edge cost. The default is 'weight'.

maxtime : float or None, optional
Specifies the maximum amount of time to allow for computing the minimum-cost network flow. The default is None.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None

Returns a graph that has the edge attributes *flow* and *reduced_cost*, which contain the flow and reduced cost values, respectively; and the node attribute *dual*, which contains the optimal dual value. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned.

9

Matching Algorithms

Functions

Function Name	Description
minimum_weight_full_matching	Returns a minimum-weight full matching of the bipartite graph *g*.

minimum_weight_full_matching

Returns a minimum-weight full matching of the bipartite graph *g*.

API: minimum_weight_full_matching

```
sasviya.network.algorithms.matching.minimum_weight_full_matching(g,
edge_weight=None, *, data=True)
```

Parameters

g : SAS graph object
Specifies the input bipartite directed graph to assess.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns a directed graph whose edges identify the assignment mapping in the minimum-weight full matching solution. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

10

Minimum Cut Algorithms

Functions

Function Name	Description
min_cut	Calculates a minimum cut of a graph.
min_s_t_cut	Calculates a minimum s-t cut of a graph.

min_cut

Calculates a minimum cut of a graph.

API: min_cut

```
sasviya.network.algorithms.min_cuts.min_cut(g, edge_weight=None, *,
inplace=False, data=True)
```

Parameters

g : SAS graph object
Specifies the simple undirected input graph for which to calculate a minimum cut.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted minimum cut is calculated.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute 'mincut', which indicates the edges to include in the minimum cut set, and the node attribute 'mincut_partition', which indicates the node partition of the calculated minimum cut. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, a modified copy of the graph is returned.

min_s_t_cut

Calculates a minimum s-t cut of a graph.

API: min_s_t_cut

```
sasviya.network.algorithms.min_cuts.min_s_t_cut(g, source, sink,
edge_weight=None, *, inplace=False, data=True)
```

Parameters

g : SAS graph object
Specifies the input graph for which to calculate a minimum s-t cut.

source : number or str
Specifies the source node (s) for the minimum s-t cut.

sink : number or str
Specifies the sink node (t) for the minimum s-t cut.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted minimum s-t cut is calculated.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute *s_t_cut*, which indicates the edges to include in the minimum s-t cut set; and the node attribute *s_t_cut_partition*, which indicates the node partition of the calculated minimum s-t cut. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, a modified copy of the graph is returned.

11

Node Similarity Algorithms

Functions

Function Name	Description
similarity_adamic_adar	Computes Adamic-Adar node similarity.
similarity_common_neighbors	Computes common neighbors node similarity.
similarity_cosine	Computes cosine node similarity.
similarity_dice	Computes Sørensen-Dice node similarity.
similarity_graph	Computes a node similarity graph.
similarity_graph_adamic_adar	Computes an Adamic-Adar node similarity graph.
similarity_graph_common_neighbors	Computes a common neighbors node similarity graph.
similarity_graph_cosine	Computes a cosine node similarity graph.
similarity_graph_dice	Computes a Sørensen-Dice node similarity graph.
similarity_graph_jaccard	Computes a Jaccard node similarity graph.
similarity_jaccard	Computes Jaccard node similarity.

similarity_adamic_adar

Computes Adamic-Adar node similarity.

API: similarity_adamic_adar

```
sasviya.network.algorithms.node_similarity.similarity_adamic_adar(g, source,
sink=None, top_k=None, bottom_k=None, *, exclude_source=False,
exclude_existing_neighbors=False)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_common_neighbors

Computes common neighbors node similarity.

API: similarity_common_neighbors

```
sasviya.network.algorithms.node_similarity.similarity_common_neighbors(g,
source, sink=None, top_k=None, bottom_k=None, *, exclude_source=False,
exclude_existing_neighbors=False)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_cosine

Computes cosine node similarity.

API: similarity_cosine

```
sasviya.network.algorithms.node_similarity.similarity_cosine(g, source,
sink=None, top_k=None, bottom_k=None, *, exclude_source=False,
exclude_existing_neighbors=False)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_dice

Computes Sørensen-Dice node similarity.

API: similarity_dice

```
sasviya.network.algorithms.node_similarity.similarity_dice(g, source, sink=None,
top_k=None, bottom_k=None, *, exclude_source=False,
exclude_existing_neighbors=False)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_graph

Computes a node similarity graph.

API: similarity_graph

```
sasviya.network.algorithms.node_similarity.similarity_graph(g, measure,
min_score=2.220446049250313e-16, max_score=inf, *, exclude_source=False,
exclude_existing_neighbors=False, data=True, **kwargs)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

measure : str, one of {'adamic_adar', 'common_neighbors', 'cosine', 'dice', 'jaccard'}
Specifies the metric to be used to compute the node similarity.

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the edge attribute that corresponds to the measure name. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_adamic_adar

Computes an Adamic-Adar node similarity graph.

API: similarity_graph_adamic_adar

sasviya.network.algorithms.node_similarity.similarity_graph_adamic_adar(g, min_score=2.220446049250313e-16, max_score=inf, *, exclude_source=False, exclude_existing_neighbors=False, data=True)

Parameters

g : SAS graph object
Specifies the input graph to assess.

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'adamic_adar' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_common_neighbors

Computes a common neighbors node similarity graph.

API: similarity_graph_common_neighbors

sasviya.network.algorithms.node_similarity.similarity_graph_common_neighbors(g, min_score=2.220446049250313e-16, max_score=inf, *, exclude_source=False, exclude_existing_neighbors=False, data=True)

Parameters

g : SAS graph object
Specifies the input graph to assess.

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'common_neighbors' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_cosine

Computes a cosine node similarity graph.

API: similarity_graph_cosine

```
sasviya.network.algorithms.node_similarity.similarity_graph_cosine(g,
edge_weight=None, min_score=2.220446049250313e-16, max_score=inf, *,
exclude_source=False, exclude_existing_neighbors=False, data=True)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'cosine' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_dice

Computes a Sørensen-Dice node similarity graph.

API: similarity_graph_dice

```
sasviya.network.algorithms.node_similarity.similarity_graph_dice(g,
min_score=2.220446049250313e-16, max_score=inf, *, exclude_source=False,
exclude_existing_neighbors=False, data=True)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the *dice* edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_jaccard

Computes a Jaccard node similarity graph.

API: similarity_graph_jaccard

```
sasviya.network.algorithms.node_similarity.similarity_graph_jaccard(g,
min_score=2.220446049250313e-16, max_score=inf, *, exclude_source=False,
exclude_existing_neighbors=False, data=True)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'jaccard' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_jaccard

Computes Jaccard node similarity.

API: similarity_jaccard

```
sasviya.network.algorithms.node_similarity.similarity_jaccard(g, source,
sink=None, top_k=None, bottom_k=None, *, exclude_source=False,
exclude_existing_neighbors=False)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

12

Path Algorithms

Functions

Function Name	Description
enumerate_path_edges	Enumerates the paths in a graph and returns the edges of the paths.
enumerate_path_nodes	Enumerates the paths in a graph and returns the nodes of the paths.
enumerate_paths	Enumerates the paths in a graph and returns the Path graph objects.

enumerate_path_edges

Enumerates the paths in a graph and returns the edges of the paths.

API: enumerate_path_edges

```
sasviya.network.algorithms.paths.enumerate_path_edges(g, source=None,
sink=None, minlength=1, maxlength=None, edge_weight=None,
node_weight=None, minedgewt=None, maxedgewt=None, minnodewt=None,
maxnodewt=None, maxtime=None)
```

Parameters

g : SAS graph object
Specifies the input graph for which to enumerate paths.

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

Yields

generator
Yields a generator of Path objects, each of which is given as a list of the paths' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_path_nodes

Enumerates the paths in a graph and returns the nodes of the paths.

API: enumerate_path_nodes

```
sasviya.network.algorithms.paths.enumerate_path_nodes(g, source=None,
sink=None, minlength=1, maxlength=None, edge_weight=None,
node_weight=None, minedgewt=None, maxedgewt=None, minnodewt=None,
maxnodewt=None, maxtime=None)
```

Parameters

g : SAS graph object
Specifies the input graph for which to enumerate paths.

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

Yields

generator
Yields a generator of Path objects, each of which is given as a list of the paths' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_paths

Enumerates the paths in a graph and returns the Path graph objects.

API: enumerate_paths

```
sasviya.network.algorithms.paths.enumerate_paths(g, source=None, sink=None,
minlength=1, maxlength=None, edge_weight=None, node_weight=None,
```

minedgewt=None, maxedgewt=None, minnodewt=None, maxnodewt=None, maxtime=None, *, data=True)

Parameters

g : SAS graph object
Specifies the input graph for which to enumerate paths.

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one Path object. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

13

Projection Algorithms

Functions

Function Name	Description
projected_graph	Computes a projected graph.
projected_graph_adamic_adar	Computes an Adamic-Adar projected graph.
projected_graph_common_neighbors	Computes a common neighbors projected graph.
projected_graph_cosine	Computes a cosine projected graph.
projected_graph_dice	Computes a Sørensen-Dice projected graph.
projected_graph_jaccard	Computes a Jaccard projected graph.

projected_graph

Computes a projected graph.

API: projected_graph

sasviya.network.algorithms.projection.projected_graph(g, nodes, neighbors=None, directed_method=None, _measure=None, *, multigraph=False, data=True)

The algorithm assumes that the graph that is induced by union(*nodes*, *neighbors*) in *g* is bipartite, so any edges that exist internally for the partitions that are defined by *nodes* or *neighbors* are ignored.

Parameters

g : SAS graph object
Specifies the input graph to assess.

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than *nodes*.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used. When the *directed_method* parameter is specified for an undirected graph, a ValueError is raised.

multigraph : bool, optional
Specifies whether or not to return a multigraph. When the value is True, the algorithm returns a multigraph in which the multiple edges represent multiple shared neighbors. The edge attribute 'neighbor' stores the label of the shared neighbor. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

projected_graph_adamic_adar

Computes an Adamic-Adar projected graph.

API: projected_graph_adamic_adar

sasviya.network.algorithms.projection.projected_graph_adamic_adar(g, nodes, neighbors=None, directed_method=None, *, data=True)

Parameters

g : SAS graph object
Specifies the input graph to assess.

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than *nodes*.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'adamic_adar' edge attribute.

projected_graph_common_neighbors

Computes a common neighbors projected graph.

API: projected_graph_common_neighbors

```
sasviya.network.algorithms.projection.projected_graph_common_neighbors(g,
nodes, neighbors=None, directed_method=None, *, data=True)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'common_neighbors' edge attribute.

projected_graph_cosine

Computes a cosine projected graph.

API: projected_graph_cosine

```
sasviya.network.algorithms.projection.projected_graph_cosine(g, nodes,
edge_weight=None, neighbors=None, directed_method=None, *, data=True)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

nodes : list or iterable
Specifies the subset of nodes to project onto.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'cosine' edge attribute.

projected_graph_dice

Computes a Sørensen-Dice projected graph.

API: projected_graph_dice

sasviya.network.algorithms.projection.projected_graph_dice(g, nodes, neighbors=None, directed_method=None, *, data=True)

Parameters

g : SAS graph object
Specifies the input graph to assess.

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the subset of nodes that are specified by the *neighbors* parameter. The strength of association in the calculated projection is represented by the 'dice' edge attribute.

projected_graph_jaccard

Computes a Jaccard projected graph.

API: projected_graph_jaccard

```
sasviya.network.algorithms.projection.projected_graph_jaccard(g, nodes,
neighbors=None, directed_method=None, *, data=True)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'jaccard' edge attribute.

14

Query Algorithms

Functions

Function Name	Description
query	Queries for the subgraph isomorphisms of q within g.

query

Queries for the subgraph isomorphisms of q within g.

API: query

```
sasviya.network.algorithms.query.query(g, q, expr=None,
_node_attr_to_ignore=None, _edge_attr_to_ignore=None, *, induced=False,
break_symmetry=False, _generate_output=True, _orbits_only=False, **kwargs)
```

Query results are optionally subject to additional filters that are represented by the *expr* parameter.

Parameters

g : graph
Specifies the main graph.

q : graph
Specifies the query graph.

expr : _Expression or None, optional
Specifies additional logic to filter the query results. The default is None.

induced : bool, optional
Specifies whether or not to return only node-induced matches. The default is False.

break_symmetry : bool, optional
Specifies whether or not to break symmetry and eliminate automorphic relationships between matches. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one isomorphic mapping from *q* to *g*. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

15

Reach Algorithms

Functions

Function Name	Description
reach_graph	Computes the reach graph for the source or set of sources.
reach_graph_per_source	Computes a separate reach graph for each given source node.

reach_graph

Computes the reach graph for the source or set of sources.

API: reach_graph

```
sasviya.network.algorithms.reach.reach_graph(g, source, max_reach=1, *,
data=True)
```

This is the induced subgraph over the set of nodes that are reachable within a given number of steps (or hops) from a set of source nodes. Reach networks are often referred to as ego networks in the context of social networks.

Parameters

g : SAS graph object
Specifies the input graph to assess.

source : node or iterable of nodes
Specifies one or more source nodes from which to compute the reach network.

max_reach : int, optional
Specifies the number of steps (or hops) to traverse from the source nodes. The default is 1.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns a graph that represents the reach network. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

reach_graph_per_source

Computes a separate reach graph for each given source node.

API: reach_graph_per_source

```
sasviya.network.algorithms.reach.reach_graph_per_source(g, source,
max_reach=1, *, data=True)
```

This is the induced subgraph of the set of nodes that are reachable within a given number of steps (or hops) to that source node. Reach networks are often referred to as ego networks in the context of social networks.

Parameters

g : SAS graph object
Specifies the input graph to assess.

source : node or iterable of nodes
Specifies one or more source nodes from which to compute the reach network.

max_reach : int, optional
Specifies the number of steps (or hops) to traverse from the source nodes. The default is 1.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one reach network. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

16

Routing Algorithms

Functions

Function Name	Description
vrp	Solves the vehicle routing problem.
vrptw	Solves the vehicle routing problem with time windows.

vrp

Solves the vehicle routing problem.

API: vrp

```
sasviya.network.algorithms.routing.vrp(g, capacity, depot,
node_demand='demand', edge_cost='weight', minroutes=1, maxroutes=None,
maxtime=None, minobjective=None, *, use_milp=True, data=True, **options)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

capacity : float
Specifies the capacity of each vehicle.

depot : node
Specifies the depot node for the vehicle routing problem.

node_demand : str or dict or iterable
Specifies the quantity of goods to deliver to or pick up from a customer. The default is 'demand'.

edge_cost : str or dict or iterable
Specifies the cost of traversing each edge. The default is 'weight'.

minroutes : int, optional
Specifies the minimum number of routes that are allowed to service demand.

maxroutes : int, optional
Specifies the maximum number of routes that are allowed to service demand.

maxtime : float or None, optional
Specifies the maximum amount of time for solving the vehicle routing problem. The default is None.

minobjective : float, optional
Specifies a minimum value such that when a solution is found whose objective is less than or equal to this value, the algorithm stops.

use_milp : bool, optional
Specifies whether or not to use a mixed integer linear programming (MILP) solver to solve the vehicle routing problem. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Cycle objects; each cycle represents one route. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

vrptw

Solves the vehicle routing problem with time windows.

API: vrptw

```
sasviya.network.algorithms.routing.vrptw(g, capacity, depot,
node_demand='demand', edge_cost='weight', travel_time='weight',
timewindow_lower=None, timewindow_upper=None, service_time=None,
minroutes=1, maxroutes=None, maxtime=None, *, data=True)
```

Parameters

g : SAS graph object
Specifies the simple input graph to assess.

capacity : float
Specifies the capacity of each vehicle.

depot : node
Specifies the depot node for the vehicle routing problem.

node_demand : str or dict or iterable
Specifies the quantity of goods to deliver to or pick up from a customer. The default is 'demand'.

edge_cost : str or dict or iterable
Specifies the cost of traversing each edge.

travel_time : str or dict or iterable
Specifies attributes or values that define the time that it takes to traverse each edge. The default is 'weight'.

timewindow_lower : str or dict or iterable or None, optional
Specifies attributes or values that define the lower time limits within which a delivery vehicle can arrive at the customer nodes. The default is None.

timewindow_upper : str or dict or iterable or None, optional
Specifies attributes or values that define the upper time limits within which a delivery vehicle can arrive at the customer nodes. The default is None.

service_time : str or dict or iterable or None, optional
Specifies attributes or values that define the service time of the nodes. The default is None.

minroutes : int, optional
Specifies the minimum number of routes that are allowed to service demand.

maxroutes : int, optional
Specifies the maximum number of routes that are allowed to service demand.

maxtime : float or None, optional
Specifies the maximum amount of time for solving the vehicle routing problem. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Cycle objects; each cycle represents one route. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

17

Shortest Path Algorithms

Functions

Function Name	Description
enumerate_sequence_shortest_paths	Enumerates the shortest paths in a graph that visit the indicated sequence nodes in order.
enumerate_shortest_path_edges	Enumerates the shortest paths in a graph and returns the edges of the paths.
enumerate_shortest_path_nodes	Enumerates the shortest paths in a graph and returns the nodes of the paths.
enumerate_shortest_paths	Enumerates the shortest paths in a graph and returns the Path graph objects.
shortest_path_distances	Calculates the shortest path distances in a graph.

enumerate_sequence_shortest_paths

Enumerates the shortest paths in a graph that visit the indicated sequence nodes in order.

API: enumerate_sequence_shortest_paths

sasviya.network.algorithms.shortest_paths.enumerate_sequence_shortest_paths(g, sequence, pathsperpair=1, edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None, maxrelobjgap=None, *, data=True)

Parameters

g : SAS graph object
Specifies the input graph to assess.

sequence : iterable of nodes
Specifies the order in which to visit the required nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Path objects; each object represents a subpath between a sequence of nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_path_edges

Enumerates the shortest paths in a graph and returns the edges of the paths.

API: enumerate_shortest_path_edges

```
sasviya.network.algorithms.shortest_paths.enumerate_shortest_path_edges(g,
source=None, sink=None, pathsperpair=1, edge_weight=None,
minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None)
```

Parameters

g : SAS graph object
Specifies the input graph for which to enumerate shortest paths.

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

Yields

generator
Yields a generator of paths; each path is given as a list of the paths' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_path_nodes

Enumerates the shortest paths in a graph and returns the nodes of the paths.

API: enumerate_shortest_path_nodes

```
sasviya.network.algorithms.shortest_paths.enumerate_shortest_path_nodes(g,
source=None, sink=None, pathsperpair=1, edge_weight=None,
minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None)
```

Parameters

g : SAS graph object
Specifies the input graph for which to enumerate shortest paths.

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

Yields

generator
Yields a generator of paths; each path is given as a list of the paths' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_paths

Enumerates the shortest paths in a graph and returns the Path graph objects.

API: enumerate_shortest_paths

```
sasviya.network.algorithms.shortest_paths.enumerate_shortest_paths(g,
source=None, sink=None, pathsperpair=1, edge_weight=None,
minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None, *, data=True)
```

Parameters

g : SAS graph object
Specifies the input graph for which to enumerate shortest paths.

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Path objects; each object represents one shortest path. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

shortest_path_distances

Calculates the shortest path distances in a graph.

API: shortest_path_distances

```
sasviya.network.algorithms.shortest_paths.shortest_path_distances(g,
source=None, sink=None, edge_weight=None, minedgewt=None,
maxedgewt=None)
```

Parameters

g : SAS graph object
Specifies the input graph for which to enumerate shortest paths.

source : node or iterable of nodes or None, optional
Specifies the source node(s) and restricts enumeration to the shortest paths that contain the specified source node(s). The default is None.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) and restricts enumeration to the shortest paths that contain the specified sink node(s). The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

Returns

dict
Returns a dictionary where each key is a (source, sink) tuple and its value is the corresponding shortest source-sink path distance.

18

Spanning Tree Algorithms

Functions

Function Name	Description
minimum_spanning_tree	Calculates a minimum spanning tree (MST) or forest of a graph.

minimum_spanning_tree

Calculates a minimum spanning tree (MST) or forest of a graph.

API: minimum_spanning_tree

```
sasviya.network.algorithms.spanning_trees.minimum_spanning_tree(g,
edge_weight=None, source=None, *, inplace=False, data=True)
```

Parameters

g : SAS graph object
Specifies the input graph for which to calculate an MST or forest.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

source : number or str or None
Specifies the source node for a directed MST. The default is None.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute 'mst', which indicates the edges to include in the spanning tree or forest. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, the returned graph contains only the spanning tree or forest edges.

19

Summary Algorithms

Functions

Function Name	Description
approximate_diameter	Computes an approximation of the graph's diameter.
articulation_point_count	Computes a graph's articulation point count.
assortativity_degree	Computes a graph's degree assortativity.
assortativity_nominal	Computes a graph's nominal assortativity.
assortativity_numeric	Computes a graph's numeric assortativity.
average_shortest_path	Computes a graph's average shortest path.
biconnected_component_count	Computes a graph's biconnected component count.
connected_component_count	Computes a graph's connected component count.
diameter	Computes a graph's diameter.
leaf_node_count	Computes a graph's leaf node count.
singleton_node_count	Computes a graph's singleton node count.
triangle_count	Computes a graph's triangle count.

approximate_diameter

Computes an approximation of the graph's diameter.

API: approximate_diameter

```
sasviya.network.algorithms.summary.approximate_diameter(g,
edge_weight=None)
```

Parameters

g : SAS graph object
Specifies the undirected input graph to assess.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted approximate diameter is computed.

Returns

float
Returns the approximate diameter.

articulation_point_count

Computes a graph's articulation point count.

API: articulation_point_count

```
sasviya.network.algorithms.summary.articulation_point_count(g)
```

Parameters

g : SAS graph object
Specifies the undirected input graph to assess.

Returns

int
Returns the number of articulation points.

assortativity_degree

Computes a graph's degree assortativity.

API: assortativity_degree

```
sasviya.network.algorithms.summary.assortativity_degree(g,
source_direction='out', target_direction='in', edge_weight=None)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

source_direction : {'in', 'out'}
Specifies the degree type (in-degree or out-degree) for the source node in a directed graph.

target_direction : {'in', 'out'}
Specifies the degree type (in-degree or out-degree) for the target node in a directed graph.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted assortativity degree is computed.

Returns

float
Returns the degree assortativity.

assortativity_nominal

Computes a graph's nominal assortativity.

API: assortativity_nominal

```
sasviya.network.algorithms.summary.assortativity_nominal(g, node_attr)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

node_attr : str
Specifies the name of the nominal node attribute to use.

Returns

float
Returns the nominal assortativity.

assortativity_numeric

Computes a graph's numeric assortativity.

API: assortativity_numeric

```
sasviya.network.algorithms.summary.assortativity_numeric(g, node_attr)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

node_attr : str
Specifies the name of the numeric node attribute to use.

Returns

float
Returns the numeric assortativity.

average_shortest_path

Computes a graph's average shortest path.

API: average_shortest_path

```
sasviya.network.algorithms.summary.average_shortest_path(g,
edge_weight=None)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted average shortest path is computed.

Returns

float
Returns the average shortest path.

biconnected_component_count

Computes a graph's biconnected component count.

API: biconnected_component_count

```
sasviya.network.algorithms.summary.biconnected_component_count(g)
```

Parameters

g : SAS graph object
Specifies the undirected input graph to assess.

Returns

int
Returns the number of biconnected components.

connected_component_count

Computes a graph's connected component count.

API: connected_component_count

```
sasviya.network.algorithms.summary.connected_component_count(g)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

Returns

int
Returns the number of connected components.

diameter

Computes a graph's diameter.

API: diameter

```
sasviya.network.algorithms.summary.diameter(g, edge_weight=None)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted diameter is computed.

Returns

int or float
Returns the diameter.

leaf_node_count

Computes a graph's leaf node count.

API: leaf_node_count

```
sasviya.network.algorithms.summary.leaf_node_count(g)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

Returns

int
Returns the number of leaf nodes.

singleton_node_count

Computes a graph's singleton node count.

API: singleton_node_count

```
sasviya.network.algorithms.summary.singleton_node_count(g)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

Returns

int
Returns the number of singleton nodes.

triangle_count

Computes a graph's triangle count.

API: triangle_count

```
sasviya.network.algorithms.summary.triangle_count(g)
```

Parameters

g : SAS graph object
Specifies the undirected input graph to assess.

Returns

int
Returns the number of triangles.

20

Topological Sort Algorithms

Functions

Function Name	Description
add_topological_order_attr	Computes a topological ordering and stores the results as node attributes of the graph.
topological_ordering	Finds a topological ordering of the graph's nodes.

add_topological_order_attr

Computes a topological ordering and stores the results as node attributes of the graph.

API: add_topological_order_attr

sasviya.network.algorithms.topological_sort.add_topological_order_attr(g, *, inplace=False, data=True)

Parameters

g : directed SAS graph object
Specifies the directed input graph for which to calculate the topological ordering.

inplace : bool, optional
Specifies whether or not to add the topological order values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a directed graph whose topological order is stored as a node attribute named *top_order*. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

topological_ordering

Finds a topological ordering of the graph's nodes.

API: topological_ordering

```
sasviya.network.algorithms.topological_sort.topological_ordering(g)
```

Parameters

g : directed SAS graph object
Specifies the directed input graph for which to calculate the topological ordering.

Yields

generator
Yields a generator of nodes in topological order. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

21

Traveling Salesman Algorithms

Functions

Function Name	Description
is_hamiltonian	Determines whether the graph is Hamiltonian (has a Hamiltonian cycle) or not.
tsp_tour	Calculates a traveling salesman problem (TSP) tour of a graph.

is_hamiltonian

Determines whether the graph is Hamiltonian (has a Hamiltonian cycle) or not.

API: is_hamiltonian

```
sasviya.network.algorithms.tsp.is_hamiltonian(g, maxtime=None)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

maxtime : float or None, optional
Specifies the maximum amount of time to determine whether or not the graph is Hamiltonian. The default is None.

Returns

bool or None
Indicates whether or not the input graph is determined to be Hamiltonian. Returns None if no determination is made.

tsp_tour

Calculates a traveling salesman problem (TSP) tour of a graph.

API: tsp_tour

```
sasviya.network.algorithms.tsp.tsp_tour(g, edge_weight=None, maxtime=None,
minobjective=None, *, inplace=False, use_milp=True, data=True, **options)
```

Parameters

g : SAS graph object
Specifies the input graph for which to calculate a TSP tour.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted TSP is calculated.

maxtime : float or None, optional
Specifies the maximum amount of time for solving a TSP. The default is None.

minobjective : float, optional
Specifies a minimum value such that when a solution is found whose objective is less than or equal to this value, the algorithm stops.

inplace : bool, optional
Specifies whether to add the TSP order values as attributes to the input graph directly or to create a copy of the graph. The default is False.

use_milp : bool, optional
Specifies whether or not to use a mixed integer linear programming (MILP) solver to solve a TSP. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is False, the returned graph is a Cycle that contains the calculated TSP tour edges. When the *inplace* parameter value is True, the graph is modified in place and None is returned. The modified graph contains the edge attributes *tsp* and *tsp_order* and the node attribute *tsp_order*.

22

Mutable Graph Classes

Classes

Class Name	Description
DiGraph	Base class for directed graphs.
Graph	Base class for undirected graphs.
MultiDiGraph	Base class for directed multigraphs.
MultiGraph	Base class for undirected multigraphs.

DiGraph

Base class for directed graphs.

API: DiGraph

class sasviya.network.classes.digraph.DiGraph(*args, **kwargs)

Self-loops and multiedges are not allowed.

Attributes

edges
Returns the edges of the graph.

nodes
Returns the nodes of the graph.

Methods

Method Name	Description
add_core_number_attr	Computes the core numbers and stores the results as node attributes on the graph.
add_edge	Adds a single edge to the graph.
add_edges_from	Adds specified edges to the graph.
add_node	Adds a node to the graph.
add_nodes_from	Adds the specified nodes to the graph.
add_topological_order_attr	Computes a topological ordering and stores the results as node attributes of the graph.
add_weighted_edges_from	Adds weighted edges to the graph.
adjacency	Returns an iterator over (node, adjacency dict) for all nodes.
allows_multiedges	Specifies whether or not the graph allows multiedges.
allows_selfloops	Specifies whether or not the graph allows self-loops.
apply	Applies a function to all node or edge attributes and updates them.
approximate_diameter	Computes an approximation of the graph's diameter.
articulation_point_count	Computes a graph's articulation point count.
articulation_points	Finds the articulation points (cut points) of a graph.
assortativity_degree	Computes a graph's degree assortativity.

Method Name	Description
assortativity_nominal	Computes a graph's nominal assortativity.
assortativity_numeric	Computes a graph's numeric assortativity.
average_shortest_path	Computes a graph's average shortest path.
biconcomp_distribution_edges	Computes the distribution of the number of edges in each biconnected components by using *numpy.histogram*.
biconcomp_distribution_nodes	Computes the distribution of the number of nodes in each biconnected components by using *numpy.histogram*.
biconcomp_size_edges	Computes the number of edges within each biconnected component.
biconcomp_size_nodes	Computes the number of nodes within each biconnected component.
biconnected_component_count	Computes a graph's biconnected component count.
biconnected_components	Computes the graph's biconnected components for an undirected graph.
block_cut_tree	Computes the block-cut tree of a graph.
centrality_betweenness	Computes weighted or unweighted betweenness centrality and stores the results as attributes on the graph.
centrality_closeness	Computes weighted or unweighted closeness centrality and stores the results as attributes on the graph.
centrality_degree	Computes weighted or unweighted degree centrality and stores the results as attributes on the graph.
centrality_eigenvector	Computes weighted or unweighted eigenvector centrality and stores the results as attributes on the graph.
centrality_hub_authority	Computes weighted or unweighted hub and authority centrality and stores the results as attributes on the graph.
centrality_influence	Computes weighted or unweighted influence centrality and stores the results as attributes on the graph.

Method Name	Description
centrality_pagerank	Computes weighted or unweighted PageRank centrality and stores the results as attributes on the graph.
clear	Removes all nodes and edges from the graph.
clear_edges	Removes all edges from the graph but retains nodes.
concomp_distribution_edges	Computes the distribution of the number of edges in each connected component by using *numpy.histogram*.
concomp_distribution_nodes	Computes the distribution of the number of nodes in each connected component by using *numpy.histogram*.
concomp_size_edges	Computes the number of edges within each connected component.
concomp_size_nodes	Computes the number of nodes within each connected component.
connected_component_count	Computes a graph's connected component count.
connected_components	Computes the connected components of a graph.
copy	Returns a copy of the input graph.
core_decomposition	Computes a core decomposition of the graph.
cycle_count	Counts the number of cycles in a graph.
degree	Returns the degree of nodes.
density	Computes the density of the graph.
diameter	Computes a graph's diameter.
drop_edge_attributes	Removes edge attributes if they exist.
drop_node_attributes	Removes node attributes if they exist.
enumerate_cycle_edges	Enumerates the cycles in a graph and returns the edges of the cycles.
enumerate_cycle_nodes	Enumerates the cycles in a graph and returns the nodes of the cycles.
enumerate_cycles	Enumerates the cycles in a graph and returns the cycle graph objects.

Method Name	Description
enumerate_path_edges	Enumerates the paths in a graph and returns the edges of the paths.
enumerate_path_nodes	Enumerates the paths in a graph and returns the nodes of the paths.
enumerate_paths	Enumerates the paths in a graph and returns the Path graph objects.
enumerate_sequence_shortest_paths	Enumerates the shortest paths in a graph that visit the indicated sequence nodes in order.
enumerate_shortest_path_edges	Enumerates the shortest paths in a graph and returns the edges of the paths.
enumerate_shortest_path_nodes	Enumerates the shortest paths in a graph and returns the nodes of the paths.
enumerate_shortest_paths	Enumerates the shortest paths in a graph and returns the Path graph objects.
get_meta_graph	Constructs a metagraph from the input graph.
get_nlargest_biconcomps	Returns a list of up to *n* largest biconnected components.
get_nlargest_concomps	Returns a list of up to *n* largest connected (with highest number of nodes) components of a graph.
has_edge_attribute	Checks the edges of the input graph for the specified edge attribute.
has_node_attribute	Checks the nodes of the input graph for the specified node attribute.
in_degree	Returns the in-degree of nodes.
intersection	Returns the intersection of input graphs by finding the intersections of their nodes and edges.
is_biconnected	Specifies whether or not the input graph is biconnected.
is_bipartite	Determines whether the input graph is bipartite.
is_connected	Specifies whether or not the input graph is connected.
is_cycle	Determines whether the input graph is a simple cycle.

Method Name	Description
is_dag	Determines whether the input graph is a directed acyclic graph (DAG).
is_directed	Specifies whether or not the graph is directed.
is_hamiltonian	Determines whether the graph is Hamiltonian (has a Hamiltonian cycle) or not.
is_immutable	Specifies whether or not the graph is immutable.
is_multigraph	Specifies whether or not the graph allows multiedges.
is_path	Determines whether the input graph is a simple path.
is_strongly_connected	Specifies whether the input graph is strongly connected.
is_weakly_connected	Specifies whether the input graph is weakly connected.
label_propagation	Computes the communities of the graph by using the label propagation algorithm.
label_propagation_nodes	Computes community assignments for all nodes in the graph by using the label propagation algorithm.
leaf_node_count	Computes a graph's leaf node count.
local_transitivity	Computes the local transitivity (clustering coefficient) and stores the results as node attributes on the graph.
louvain	Computes the communities of the graph by using the Louvain algorithm.
louvain_nodes	Computes community assignments for all nodes in the graph by using the Louvain algorithm.
maxflow	Calculates the maximum flow from a source node to a sink node in a graph.
maximal_cliques	Enumerates all maximal cliques of the graph.
min_cut	Calculates a minimum cut of a graph.
min_s_t_cut	Calculates a minimum s-t cut of a graph.
mincostflow	Calculates the minimum-cost network flow for the graph.

Method Name	Description
minimum_spanning_tree	Calculates a minimum spanning tree (MST) or forest of a graph.
minimum_weight_full_matching	Returns a minimum-weight full matching of the bipartite graph *self*.
neighbors	Returns an iterator over all neighbors of node n.
node_clique_numbers	Computes node clique numbers and stores the results as attributes on the graph.
number_of_edges	Returns the number of edges in the graph.
number_of_nodes	Returns the number of nodes in the graph.
out_degree	Returns the out-degree of nodes.
predecessors	Returns an iterator over all predecessor nodes of n.
projected_graph	Computes a projected graph.
projected_graph_adamic_adar	Computes an Adamic-Adar projected graph.
projected_graph_common_neighbors	Computes a common neighbors projected graph.
projected_graph_cosine	Computes a cosine projected graph.
projected_graph_dice	Computes a Sørensen-Dice projected graph.
projected_graph_jaccard	Computes a Jaccard projected graph.
query	Queries for the subgraph isomorphisms of q within g.
reach_graph	Computes the reach graph for the source or set of sources.
reach_graph_per_source	Computes a separate reach graph for each given source node.
remove_edge	Removes a single edge from the graph.
remove_edges_from	Removes specified edges from the graph.
remove_isolated_nodes	Removes isolated nodes from a graph.
remove_node	Removes a single node from the graph.
remove_nodes_from	Removes specified nodes from the graph.

Method Name	Description
reverse	Returns a copy of a directed graph with reversed edges.
set_edge_attributes	Sets edge attributes by using a given value or dictionary of values.
set_node_attributes	Sets node attributes from a given value or dictionary of values.
shortest_path_distances	Calculates the shortest path distances in a graph.
similarity_adamic_adar	Computes Adamic-Adar node similarity.
similarity_common_neighbors	Computes common neighbors node similarity.
similarity_cosine	Computes cosine node similarity.
similarity_dice	Computes Sørensen-Dice node similarity.
similarity_graph	Computes a node similarity graph.
similarity_graph_adamic_adar	Computes an Adamic-Adar node similarity graph.
similarity_graph_common_neighbors	Computes a common neighbors node similarity graph.
similarity_graph_cosine	Computes a cosine node similarity graph.
similarity_graph_dice	Computes a Sørensen-Dice node similarity graph.
similarity_graph_jaccard	Computes a Jaccard node similarity graph.
similarity_jaccard	Computes Jaccard node similarity.
singleton_node_count	Computes a graph's singleton node count.
successors	Returns an iterator over all successor nodes of n.
to_directed	Returns a directed copy of the input graph.
to_immutable	Creates an immutable graph from the input graph.
to_mutable	Creates a mutable graph from the input graph.
to_pandas_edges	Returns the graph edges as a Pandas DataFrame.
to_pandas_nodes	Returns the graph nodes as a Pandas DataFrame.
to_undirected	Returns a directed copy of the input graph.

Method Name	Description
topological_ordering	Finds a topological ordering of the graph's nodes.
transitive_closure	Calculates the transitive closure of a graph.
triangle_count	Computes a graph's triangle count.
tsp_tour	Calculates a traveling salesman problem (TSP) tour of a graph.
union	Returns the union of input graphs by finding the unions of their nodes and edges.
vrp	Solves the vehicle routing problem.
vrptw	Solves the vehicle routing problem with time windows.

add_core_number_attr

```
add_core_number_attr(maxtime=None, *, inplace=False, data=True)
```

Computes the core numbers and stores the results as node attributes on the graph.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time for core decomposition to spend. The default is None.

inplace : bool, optional
Specifies whether to add the core numbers as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph with core numbers that are stored as a node attribute named *core*. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

add_edge

```
add_edge(u, v, error_on_selfloop=None, **attrs)
```

Adds a single edge to the graph.

Parameters

u : numeric or str
Specifies the *from* node.

v : numeric or str
Specifies the *to* node.

error_on_selfloop : bool or None, optional
Specifies whether or not to raise a ValueError if a self-loop is detected; that is, if (u==v). The default is None, which means that the input edge is silently ignored.

Examples

```
G.add_edge(1,2,weight=3,type='parent')
```

add_edges_from

```
add_edges_from(edges, error_on_selfloop=None, **kwargs)
```

Adds specified edges to the graph.

Parameters

edges : an iterable container of edges
Specifies the edges to add to the graph.

error_on_selfloop : bool or None, optional
Specifies whether or not to raise a ValueError if a self-loop is detected; that is, if (u==v). The default is None, which means that the input self-loop is silently ignored.

Examples

```
G.add_edges_from([("A","B",{"weight":1,"color":"blue"}),
                  ("C","D",{"weight":2,"color":"red"})])
G.add_edges_from([['X', 'Y'],["Y","Z"], weight=1,color= "blue")
```

add_node

```
add_node(node, **attrs)
```

Adds a node to the graph.

Parameters

node : str or number or tuple
Specifies the ID of the node to add to the graph. If the value is a tuple, the first element should contain the node ID, and the second element is a dictionary that represents the node's attributes.

add_nodes_from

```
add_nodes_from(nodes, **attrs)
```

Adds the specified nodes to the graph.

Parameters

nodes : an iterable container of nodes
Specifies the nodes to add to the graph.

Examples

```
G.add_nodes_from([('X', {"weight":11,"color": "blue"}), ("Y", {"color": "blue"})])
G.add_nodes_from(('X', 'Y'), weight=1,color= "blue")
G.add_nodes_from(['X', 'Y'])
```

add_topological_order_attr

```
add_topological_order_attr(*, inplace=False, data=True)
```

Computes a topological ordering and stores the results as node attributes of the graph.

Parameters

inplace : bool, optional
Specifies whether or not to add the topological order values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a directed graph whose topological order is stored as a node attribute named *top_order*. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

add_weighted_edges_from

```
add_weighted_edges_from(edges, weight='weight', **kwargs)
```

Adds weighted edges to the graph.

Parameters

edges : an iterable container of edges
Specifies the weighted edges to add to the graph.

weight : str
Specifies the attribute name for the weighted edges to add. The default is 'weight'.

adjacency

```
adjacency()
```

Returns an iterator over (node, adjacency dict) for all nodes.

For directed graphs, only outgoing adjacencies are returned.

Returns

iterator
Returns an iterator over (node, adjacency dictionary) for all nodes in the graph.

allows_multiedges

```
allows_multiedges()
```

Specifies whether or not the graph allows multiedges.

Returns

bool
Indicates whether or not the graph allows multiedges.

allows_selfloops

```
allows_selfloops()
```

Specifies whether or not the graph allows self-loops.

Returns

bool
Indicates whether or not the graph allows self-loops.

apply

```
apply(node_attr_handler=None, edge_attr_handler=None, immutable=None, *,
inplace=False)
```

Applies a function to all node or edge attributes and updates them.

#use apply to fillna >>> sasnet.apply(G, edge_attr_handler=lambda data:{'w':0 if 'w' not in data else data['w']},inplace=True) >>> assert G.edges[(1,2)]['w'] == 0 >>> print(G.nodes(data=True)) #use apply to negate an attribute >>> sasnet.apply(G, node_attr_handler=lambda data:{'x': -data['x']},inplace=True) >>> assert G.nodes[1]['x']== -10

Parameters

node_attr_handler : function or None, optional
Specifies the function to apply to node attributes. The default is None.

edge_attr_handler : function or None, optional
Specifies the function to apply to edge attributes. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the input graph. The default is False.

Returns

SAS graph object or None.
Returns the resulting graph after the function is applied, or returns nothing if the *inplace* parameter value is True.

Examples

```
import sasviya.network as sasnet
G = sasnet.Graph([[1,2,{}],[1,3,{'w':1}],[1,4,{'w':2}]])
sasnet.set_node_attributes(G,{1:10,2:20,3:30,4:40},name='x',inplace=True)
```

#use apply to fillna >>> sasnet.apply(G, edge_attr_handler=lambda data:{‘w’:0 if ‘w’ not in data else data[‘w’]},inplace=True) >>> assert G.edges[(1,2)][‘w’] == 0 >>> print(G.nodes(data=True)) #use apply to negate an attribute >>> sasnet.apply(G, node_attr_handler=lambda data:{‘x’: -data[‘x’]},inplace=True) >>> assert G.nodes[1][‘x’]== -10

approximate_diameter

approximate_diameter(edge_weight=None)

Computes an approximation of the graph’s diameter.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted approximate diameter is computed.

Returns

float
Returns the approximate diameter.

articulation_point_count

```
articulation_point_count()
```

Computes a graph's articulation point count.

Returns

int
Returns the number of articulation points.

articulation_points

```
articulation_points()
```

Finds the articulation points (cut points) of a graph.

Returns

list
Returns a list of articulation points in the graph.

assortativity_degree

```
assortativity_degree(source_direction='out', target_direction='in',
edge_weight=None)
```

Computes a graph's degree assortativity.

Parameters

source_direction : {'in', 'out'}
Specifies the degree type (in-degree or out-degree) for the source node in a directed graph.

target_direction : {'in', 'out'}
Specifies the degree type (in-degree or out-degree) for the target node in a directed graph.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted assortativity degree is computed.

Returns

float
Returns the degree assortativity.

assortativity_nominal

```
assortativity_nominal(node_attr)
```

Computes a graph's nominal assortativity.

Parameters

node_attr : str
Specifies the name of the nominal node attribute to use.

Returns

float
Returns the nominal assortativity.

assortativity_numeric

```
assortativity_numeric(node_attr)
```

Computes a graph's numeric assortativity.

Parameters

node_attr : str
Specifies the name of the numeric node attribute to use.

Returns

float
Returns the numeric assortativity.

average_shortest_path

```
average_shortest_path(edge_weight=None)
```

Computes a graph's average shortest path.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted average shortest path is computed.

Returns

float
Returns the average shortest path.

biconcomp_distribution_edges

```
biconcomp_distribution_edges(bins=None)
```

Computes the distribution of the number of edges in each biconnected components by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is int (an integer), it specifies the number of equal-width bins. When *bins* is a list, it specifies bin ranges and should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is str (a string), it specifies the method to use for calculating the bin ranges. For example, 'stone','auto','doane','fd','rice','scott','sqrt','sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of biconnected components whose edge count is within the range of the bin.

biconcomp_distribution_nodes

```
biconcomp_distribution_nodes(bins=None)
```

Computes the distribution of the number of nodes in each biconnected components by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is int (an integer), it specifies the number of equal-width bins. When *bins* is a list, it specifies bin ranges and should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is str (a string), it specifies the method to use for calculating the bin ranges. For example, 'stone','auto','doane','fd','rice','scott','sqrt','sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of biconnected components whose node count is within the range of the bin.

biconcomp_size_edges

```
biconcomp_size_edges()
```

Computes the number of edges within each biconnected component.

Returns

dict
Returns a dictionary in which the keys are biconnected component IDs and the values are the number of edges within each component.

biconcomp_size_nodes

```
biconcomp_size_nodes()
```

Computes the number of nodes within each biconnected component.

Returns

dict
Returns a dictionary in which the keys are biconnected component IDs and the values are the number of nodes within each component.

biconnected_component_count

```
biconnected_component_count()
```

Computes a graph's biconnected component count.

Returns

int
Returns the number of biconnected components.

biconnected_components

```
biconnected_components(order_by=None, *, data=True, ascending=False)
```

Computes the graph's biconnected components for an undirected graph.

Parameters

order_by : 'node' or 'edge' or None, optional
Specifies the parameter that is used to sort the components by the number of nodes or edges within each component. The default is None, which means that the components are not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Returns the result as a generator of biconnected component subgraphs. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

block_cut_tree

```
block_cut_tree(*, return_nodes_map=False)
```

Computes the block-cut tree of a graph.

Parameters

return_nodes_map : bool, optional
Specifies whether or not to also compute the block-cut tree node map. The default is False.

Returns

SAS graph object
Returns a graph that represents the block-cut tree. When *return_nodes_map = True*, the output graph has the *nodes_map* attribute, which contains a dictionary that maps its nodes to a list of nodes in the original graph. Note: If a block is empty (that is, the corresponding biconnected component contains only articulation points), that block does not appear as a key in the *nodes_map* dictionary.

centrality_betweenness

```
centrality_betweenness(edge_weight=None, sample_percent=100, *,
inplace=False, data=True)
```

Computes weighted or unweighted betweenness centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that unweighted betweenness centrality is computed.

sample_percent : int, optional
Specifies the percentage of source nodes to sample for the approximate betweenness calculation. This parameter value should be in the range (0, 100]. The default is 100.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the node and edge attributes *between_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_closeness

```
centrality_closeness(edge_weight=None, nopath='diameter', *, inplace=False,
data=True)
```

Computes weighted or unweighted closeness centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted closeness centrality is computed.

nopath : {'diameter', 'harmonic', or 'zero'}, optional
Specifies a method to use for accounting for the shortest path distance between two nodes when a path does not exist. The valid values are as follows: - 'diameter' uses the graph diameter (plus one) as the shortest path distance between disconnected nodes. - 'harmonic' uses the harmonic formula for calculating closeness centrality. - 'zero' uses zero as the shortest path distance between disconnected nodes.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the closeness centrality node attributes *centr_close_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_degree

```
centrality_degree(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted degree centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted degree centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes

of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the degree centrality node attributes *centr_degree_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_eigenvector

```
centrality_eigenvector(edge_weight=None, maxiters=10000, *, inplace=False,
data=True)
```

Computes weighted or unweighted eigenvector centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted eigenvector centrality is computed.

maxiters : int, optional
Specifies the maximum number of iterations in order to limit the amount of computation time that is spent when convergence is slow. The default is 10,000.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the degree centrality node attributes *centr_eigen_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_hub_authority

```
centrality_hub_authority(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted hub and authority centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted hub and authority centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the hub and authority centrality node attributes *centr_hub_** and *centr_auth_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_influence

```
centrality_influence(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted influence centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted influence centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the first- and second-order influence centrality node attributes *centr_influence1_** and *centr_influence2_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_pagerank

```
centrality_pagerank(edge_weight=None, alpha=0.85, tolerance=1e-09, *,
inplace=False, data=True)
```

Computes weighted or unweighted PageRank centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted PageRank centrality is computed.

alpha : float, optional
Specifies the damping parameter for the PageRank centrality (or algorithm). The default is 0.85.

tolerance : float, optional
Specifies the tolerance parameter for the PageRank centrality (or algorithm). The default is 1E-9.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the PageRank centrality node attributes *centr_pagerank_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

clear

```
clear()
```

Removes all nodes and edges from the graph.

clear_edges

```
clear_edges()
```

Removes all edges from the graph but retains nodes.

concomp_distribution_edges

```
concomp_distribution_edges(bins=None, *, strongly=True)
```

Computes the distribution of the number of edges in each connected component by using *numpy.histogram*.

Parameters

bins : int or list or str, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is an integer (int), it represents the number of equal-width bins. When *bins* is a list, it specifies bin ranges and it should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is a string (str), it specifies the method to be used for calculating the bin ranges. For example, 'stone', 'auto', 'doane', 'fd', 'rice', 'scott', 'sqrt', 'sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of components whose edge count within the range of the bin.

concomp_distribution_nodes

```
concomp_distribution_nodes(bins=None, *, strongly=True)
```

Computes the distribution of the number of nodes in each connected component by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is an integer (int), it represents the number of equal-width bins. When *bins* is a list, it specifies bin ranges and it should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is a string (str), it specifies the method to use for calculating the bin ranges. For example, 'stone', 'auto', 'doane', 'fd', 'rice', 'scott', 'sqrt', 'sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of components whose node count is within the range of the bin.

concomp_size_edges

```
concomp_size_edges(*, strongly=True)
```

Computes the number of edges within each connected component.

Parameters

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary in which the keys are connected component IDs and the values are the number of edges within each component.

concomp_size_nodes

```
concomp_size_nodes(*, strongly=True)
```

Computes the number of nodes within each connected component.

Parameters

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary in which the keys are connected component IDs and the values are the number of nodes within each component.

connected_component_count

```
connected_component_count()
```

Computes a graph's connected component count.

Returns

int
Returns the number of connected components.

connected_components

```
connected_components(order_by=None, *, strongly=True, data=True,
ascending=False)
```

Computes the connected components of a graph.

Parameters

order_by : 'node' or 'edge' or None, optional
Specifies whether to sort the components by the number of nodes or edges within each component. The default is None, which means that the components are not sorted.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one connected component. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

copy

```
copy(immutable=None, *, data=True)
```

Returns a copy of the input graph.

Parameters

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, meaning that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns an independent copy of the input.

core_decomposition

```
core_decomposition(maxtime=None, *, data=True)
```

Computes a core decomposition of the graph.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time for core decomposition to spend. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one k-core in increasing core number. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

cycle_count

```
cycle_count(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Counts the number of cycles in a graph.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Returns

int
Returns a count of cycles in the graph.

degree

```
degree(n=None, edge_weight=None)
```

Returns the degree of nodes.

Parameters

n : single node or container of nodes or None, optional
Specifies which nodes to include. The default is None, which means that the degree of every node is returned.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None.

Returns

int or dictionary
Returns the degree of specified node(s).

density

```
density()
```

Computes the density of the graph.

Returns

float
Returns the density of the graph.

diameter

```
diameter(edge_weight=None)
```

Computes a graph's diameter.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted diameter is computed.

Returns

int or float
Returns the diameter.

drop_edge_attributes

```
drop_edge_attributes(attributes_subset=None, edges_subset=None,
immutable=None, *, inplace=False)
```

Removes edge attributes if they exist.

Parameters

attributes_subset : str or list, optional
Specifies the edge attribute or a list of the edge attributes to remove; for example, 'weight' or ['weight','color']. By default, all edge attributes are removed.

edges_subset : list or iterable of edges, optional
Specifies the edges to remove attributes from; for example, [(1,2),(2,3)]. By default the attributes for all edges are removed.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by removing the specified edge attributes. If the *inplace* parameter value is False, a copy of the input graph with the modified edge attributes is returned.

drop_node_attributes

```
drop_node_attributes(attributes_subset=None, nodes_subset=None,
immutable=None, *, inplace=False)
```

Removes node attributes if they exist.

Parameters

attributes_subset : str or list, optional
Specifies the node attribute or list of node attributes to remove; for example, ['weight','color']. By default, all node attributes are removed.

nodes_subset : list or iterable of nodes, optional
Specifies the nodes to drop attributes from; for example, [1,2,3] or (1,2,3) or [1] or (1,). By default, the attributes for all nodes are removed.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by removing the specified node attributes. If the *inplace* parameter value is False, a copy of the input graph with the modified node attributes is returned.

enumerate_cycle_edges

```
enumerate_cycle_edges(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Enumerates the cycles in a graph and returns the edges of the cycles.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Yields

generator
Yields a generator of cycles, each of which is given as a list of the cycles' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_cycle_nodes

```
enumerate_cycle_nodes(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Enumerates the cycles in a graph and returns the nodes of the cycles.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Yields

generator
Yields a generator of cycles, each of which is given as a list of the cycles' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_cycles

```
enumerate_cycles(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None, *,
data=True)
```

Enumerates the cycles in a graph and returns the cycle graph objects.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one cycle of the same type as the input graph. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_path_edges

```
enumerate_path_edges(source=None, sink=None, minlength=1,
maxlength=None, edge_weight=None, node_weight=None, minedgewt=None,
maxedgewt=None, minnodewt=None, maxnodewt=None, maxtime=None)
```

Enumerates the paths in a graph and returns the edges of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

Yields

generator
Yields a generator of Path objects, each of which is given as a list of the paths' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_path_nodes

```
enumerate_path_nodes(source=None, sink=None, minlength=1,
maxlength=None, edge_weight=None, node_weight=None, minedgewt=None,
maxedgewt=None, minnodewt=None, maxnodewt=None, maxtime=None)
```

Enumerates the paths in a graph and returns the nodes of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

Yields

generator
Yields a generator of Path objects, each of which is given as a list of the paths' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_paths

```
enumerate_paths(source=None, sink=None, minlength=1, maxlength=None,
edge_weight=None, node_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, *, data=True)
```

Enumerates the paths in a graph and returns the Path graph objects.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one Path object. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_sequence_shortest_paths

```
enumerate_sequence_shortest_paths(sequence, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None, *, data=True)
```

Enumerates the shortest paths in a graph that visit the indicated sequence nodes in order.

Parameters

sequence : iterable of nodes
Specifies the order in which to visit the required nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Path objects; each object represents a subpath between a sequence of nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_path_edges

```
enumerate_shortest_path_edges(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None)
```

Enumerates the shortest paths in a graph and returns the edges of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

Yields

generator
Yields a generator of paths; each path is given as a list of the paths' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_path_nodes

```
enumerate_shortest_path_nodes(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None)
```

Enumerates the shortest paths in a graph and returns the nodes of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

Yields

generator
Yields a generator of paths; each path is given as a list of the paths' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_paths

```
enumerate_shortest_paths(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None, *, data=True)
```

Enumerates the shortest paths in a graph and returns the Path graph objects.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Path objects; each object represents one shortest path. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

get_meta_graph

```
get_meta_graph(node_type, edge_type=None, immutable=None)
```

Constructs a metagraph from the input graph.

Parameters

node_type : str
Specifies the node attribute to consider in constructing the metagraph.

edge_type : str, optional
Specifies the edge attribute to consider in constructing the metagraph. By default, all connections between nodes are considered the same.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

Returns

SAS graph object
Returns the metagraph, which has an edge attribute count and determines the number of unique connections between pairs of nodes.

get_nlargest_biconcomps

```
get_nlargest_biconcomps(n, order_by='node', *, data=True, ascending=False)
```

Returns a list of up to *n* largest biconnected components.

Parameters

n : int
Specifies the number of graphs to return.

order_by : 'node' or 'edge' or None
Specifies whether to sort the biconnected components by the number of nodes or edges within those components. The default is None, which means that the biconnected components are not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. When the value is True, the smallest biconnected components are returned instead. The default is False.

Returns

list
Returns a list of n largest or smallest biconnected components of a graph.

get_nlargest_concomps

```
get_nlargest_concomps(n, order_by='node', *, strongly=True, data=True,
ascending=False)
```

Returns a list of up to *n* largest connected (with highest number of nodes) components of a graph.

Parameters

n : int
Specifies the number of graphs to return.

order_by : 'node' or 'edge'
Specifies whether to sort the component by number of nodes or edges within that component. The default is None, which means that the result is not sorted.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Returns

list
Returns a list of the n largest connected components of a graph.

has_edge_attribute

```
has_edge_attribute(name, *, all_edges=False)
```

Checks the edges of the input graph for the specified edge attribute.

Parameters

name : str
Specifies the name of the edge attribute.

all_edges : bool, optional
Specifies whether or not to check for the existence of the edge attribute in all or any of the edges. The default is False.

Returns

bool
If the all_edges parameter value is True, the value True is returned if that edge attribute exists in all edges; otherwise the value False is returned. If the all_edges parameter value is False, the value True is returned if at least one edge in the graph has that specified edge attribute.

has_node_attribute

```
has_node_attribute(name, *, all_nodes=False)
```

Checks the nodes of the input graph for the specified node attribute.

Parameters

name : str
Specifies the name of the node attribute.

all_nodes : bool, optional
Specifies whether or not to check for the existence of the node attribute in all or any of the nodes. The default is False.

Returns

bool
If the all_nodes parameter value is True, the value True is returned if that node attribute exists in all node; otherwise the value False is returned. If the all_nodes parameter value is False, the value True is returned if at least one node in the graph has that specified node attribute.

in_degree

```
in_degree(n=None, edge_weight=None)
```

Returns the in-degree of nodes.

Parameters

n : single node or container of nodes or None, optional
Specifies which nodes to include. The default is None, which means that the in-degree of all nodes is returned.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None.

Returns

numeric or dictionary
Returns the in-degree of the node if a single node is passed, or returns a dictionary with nodes as keys and their in-degrees as values if None or a list of nodes is passed.

intersection

```
intersection(rhs, nodes_attrs_agg=None, edges_attrs_agg=None)
```

Returns the intersection of input graphs by finding the intersections of their nodes and edges.

The node attributes and edge attributes are aggregated using the rules that are defined by the *nodes_attrs_agg* and *edges_attrs_agg* parameters.

Parameters

rhs : SAS graph object
Specifies the other graph, of the same type as *self*, with which to intersect.

nodes_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate node attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps node attributes to functions. The default is None.

edges_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate edge attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps edge attributes to functions. The default is None.

Returns

SAS graph object
Returns a graph that is constructed by the intersection of the two graphs.

is_biconnected

```
is_biconnected()
```

Specifies whether or not the input graph is biconnected.

Returns

bool
Indicates whether the input graph is biconnected (True) or not (False).

is_bipartite

```
is_bipartite()
```

Determines whether the input graph is bipartite.

Returns

bool
Indicates whether the input graph is bipartite.

is_connected

```
is_connected()
```

Specifies whether or not the input graph is connected.

Returns

bool
Indicates whether the input graph is connected (True) or not (False).

is_cycle

```
is_cycle()
```

Determines whether the input graph is a simple cycle.

Returns

bool
Indicates whether the input graph is a simple cycle (True) or not (False).

is_dag

```
is_dag()
```

Determines whether the input graph is a directed acyclic graph (DAG).

Returns

bool
Indicates whether the input graph is a directed acyclic graph (DAG).

is_directed

```
is_directed()
```

Specifies whether or not the graph is directed.

Returns

bool
Indicates whether or not the graph is directed.

is_hamiltonian

```
is_hamiltonian(maxtime=None)
```

Determines whether the graph is Hamiltonian (has a Hamiltonian cycle) or not.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time to determine whether or not the graph is Hamiltonian. The default is None.

Returns

bool or None
Indicates whether or not the input graph is determined to be Hamiltonian. Returns None if no determination is made.

is_immutable

```
is_immutable()
```

Specifies whether or not the graph is immutable.

Returns

bool
Indicates whether or not the graph is immutable.

is_multigraph

```
is_multigraph()
```

Specifies whether or not the graph allows multiedges.

Returns

bool
Indicates whether or not the graph allows multiedges.

is_path

```
is_path()
```

Determines whether the input graph is a simple path.

Returns

bool
Indicates whether the input graph is a simple path (True) or not (False).

is_strongly_connected

```
is_strongly_connected()
```

Specifies whether the input graph is strongly connected.

Returns

bool
Indicates whether the input graph is strongly connected (True) or not (False).

is_weakly_connected

```
is_weakly_connected()
```

Specifies whether the input graph is weakly connected.

Returns

bool
Indicates whether the input graph is weakly connected (True) or not (False).

label_propagation

```
label_propagation(edge_weight=None, maxiters=100, tolerance=0.05, *,
data=True)
```

Computes the communities of the graph by using the label propagation algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of label changes for all nodes in the graph is less than the value. The default is 0.05.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one community. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

label_propagation_nodes

```
label_propagation_nodes(edge_weight=None, maxiters=100, tolerance=0.05, *,
data=True, inplace=False)
```

Computes community assignments for all nodes in the graph by using the label propagation algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of label changes for all nodes in the graph is less than the value. The default is 0.05.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

inplace : bool, optional
Specifies whether to add the community values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains the node attribute *community*, which denotes the community assignment for each node. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

leaf_node_count

```
leaf_node_count()
```

Computes a graph's leaf node count.

Returns

int
Returns the number of leaf nodes.

local_transitivity

```
local_transitivity(*, inplace=False)
```

Computes the local transitivity (clustering coefficient) and stores the results as node attributes on the graph.

Parameters

inplace : bool, optional
Specifies whether to add the transitivity values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains node attribute 'transitivity'. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

louvain

```
louvain(*, edge_weight=None, resolution=1, tolerance=0.001, maxiters=100,
data=True)
```

Computes the communities of the graph by using the Louvain algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

resolution : float, optional
Specifies the factor in the modularity equation that affects the expected number of links between two nodes. A higher value produces more communities. The default is 1.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of modularity gain between two consecutive iterations is less than the value. The default is 0.001.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one community. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

louvain_nodes

```
louvain_nodes(*, edge_weight=None, resolution=1, tolerance=0.001,
maxiters=100, data=True, inplace=False)
```

Computes community assignments for all nodes in the graph by using the Louvain algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

resolution : float, optional
Specifies the factor in the modularity equation that affects the expected number of links between two nodes. A higher value produces more communities. The default is 1.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of modularity gain between two consecutive iterations is less than the value. The default is 0.001.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

inplace : bool, optional
Specifies whether to add the community values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains the node attribute *community*, which denotes the community assignment for each node. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

maxflow

```
maxflow(source, sink, capacity_attr='upper', *, inplace=False, data=True)
```

Calculates the maximum flow from a source node to a sink node in a graph.

Parameters

source : number or str
Specifies the source node for the maximum flow.

sink : number or str
Specifies the sink node for the maximum flow.

capacity_attr : str, optional
Specifies the name of the edge attribute that contains the capacity of the edge. The default is 'upper'.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute *flow*, which contains the flow values. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, the returned graph contains only the flow edges.

maximal_cliques

```
maximal_cliques(maxcliques='ALL', node_weight=None, edge_weight=None,
minsize=1, maxsize=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, order_by=None, *,
data=True, ascending=False)
```

Enumerates all maximal cliques of the graph.

Parameters

maxcliques : int or 'ALL', optional
Specifies the maximum number of enumerated cliques to return. The default is 'ALL'.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

minsize : int, optional
Specifies the minimum number of nodes in a clique. The default is 1.

maxsize : int or None, optional
Specifies the maximum number of nodes in a clique. The default is None.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a clique. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a clique. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a clique. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a clique. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds for enumeration to spend. The default is None.

order_by : 'node' or None, optional
Specifies whether to sort the cliques by the number of nodes within each clique. The default is None, which means that the result is not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one maximal clique. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

min_cut

```
min_cut(edge_weight=None, *, inplace=False, data=True)
```

Calculates a minimum cut of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted minimum cut is calculated.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute 'mincut', which indicates the edges to include in the minimum cut set, and the node attribute 'mincut_partition', which indicates the node partition of the calculated minimum cut. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, a modified copy of the graph is returned.

min_s_t_cut

```
min_s_t_cut(source, sink, edge_weight=None, *, inplace=False, data=True)
```

Calculates a minimum s-t cut of a graph.

Parameters

source : number or str
Specifies the source node (s) for the minimum s-t cut.

sink : number or str
Specifies the sink node (t) for the minimum s-t cut.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted minimum s-t cut is calculated.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute *s_t_cut*, which indicates the edges to include in the minimum s-t cut set; and the node attribute *s_t_cut_partition*, which indicates the node partition of the calculated minimum s-t cut. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, a modified copy of the graph is returned.

mincostflow

```
mincostflow(demand_lower_attr='lower', demand_upper_attr='upper',
capacity_lower_attr='lower', capacity_upper_attr='upper', cost_attr='weight',
maxtime=None, *, inplace=False, data=True)
```

Calculates the minimum-cost network flow for the graph.

Parameters

demand_lower_attr : str, optional
Specifies the name of the node attribute that contains the lower bound of the node supply. The default is 'lower'.

demand_upper_attr : str, optional
Specifies the name of the node attribute that contains the upper bound of the node supply. The default is 'upper'.

capacity_lower_attr : str, optional
Specifies the name of the edge attribute that contains the capacity lower bound of the edge. The default is 'lower'.

capacity_upper_attr : str, optional
Specifies the name of the edge attribute that contains the capacity upper bound of the edge. The default is 'upper'.

cost_attr : str, optional
Specifies the name of the edge attribute that contains the edge cost. The default is 'weight'.

maxtime : float or None, optional
Specifies the maximum amount of time to allow for computing the minimum-cost network flow. The default is None.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attributes *flow* and *reduced_cost*, which contain the flow and reduced cost values, respectively; and the node attribute *dual*, which contains the optimal dual value. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned.

minimum_spanning_tree

```
minimum_spanning_tree(edge_weight=None, source=None, *, inplace=False,
data=True)
```

Calculates a minimum spanning tree (MST) or forest of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

source : number or str or None
Specifies the source node for a directed MST. The default is None.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute 'mst', which indicates the edges to include in the spanning tree or forest. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, the returned graph contains only the spanning tree or forest edges.

minimum_weight_full_matching

```
minimum_weight_full_matching(edge_weight=None, *, data=True)
```

Returns a minimum-weight full matching of the bipartite graph *self*.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns a directed graph whose edges identify the assignment mapping in the minimum-weight full matching solution. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

neighbors

```
neighbors(n)
```

Returns an iterator over all neighbors of node n.

Parameters

n : node
Specifies a node in the graph.

Returns

iterator
Returns an iterator over all neighbors of node n.

node_clique_numbers

```
node_clique_numbers(*, inplace=False, data=True)
```

Computes node clique numbers and stores the results as attributes on the graph.

Parameters

inplace : bool, optional
Specifies whether to add the clique number values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the value of the *inplace* parameter is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph or modifies the existing graph to contain the node attribute *clique_number*, which includes the clique number of each node. The graph g has the attribute g.solution_summary, which contains information about the results of the algorithm.

number_of_edges

```
number_of_edges(u=None, v=None)
```

Returns the number of edges in the graph.

When you specify *u* and *v*, the number of edges from *u* to *v* is returned. When you specify *u* but *v* is None, the number of connections from *u* to other nodes (out-degree of node *u*) is returned. When you specify *v* but *u* is None, the number of connections from *v* to other nodes (in-degree of node *v*) is returned.

Parameters

u : numeric or str or None, optional
Specifies the *from* node. The default is None.

v : numeric or str or None, optional
Specifies the *to* node. The default is None.

Returns

int
Returns the number of edges in the graph. When you specify *u* and *v*, the number of edges from *u* to *v* is returned. When you specify *u* but *v* is None, the number of connections from *u* to other nodes(out-degree of node *u*) is returned. When you specify *v* but *u* is None, the number of connections from *v* to other nodes(in-degree of node *v*) is returned.

number_of_nodes

```
number_of_nodes()
```

Returns the number of nodes in the graph.

Returns

int
Returns the number of nodes in the graph.

out_degree

```
out_degree(n=None, edge_weight=None)
```

Returns the out-degree of nodes.

Parameters

n : single node or container of nodes or None, optional
Specifies which nodes to include. The default is None, which means that the out-degree of all nodes is returned.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None.

Returns

numeric or dictionary
Returns the out-degree of the node if a single node is passed, or returns a dictionary with nodes as keys and their out-degrees as values if None or a list of nodes is passed.

predecessors

```
predecessors(n)
```

Returns an iterator over all predecessor nodes of n.

Parameters

n : node
Specifies a node in the graph.

Returns

iterator
Returns an iterator over predecessors of node n.

projected_graph

```
projected_graph(nodes, neighbors=None, directed_method=None,
_measure=None, *, multigraph=False, data=True)
```

Computes a projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in self other than *nodes*.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used. When the *directed_method* parameter is specified for an undirected graph, a ValueError is raised.

multigraph : bool, optional
Specifies whether or not to return a multigraph. When the value is True, the algorithm returns a multigraph in which the multiple edges represent multiple shared neighbors. The edge attribute 'neighbor' stores the label of the shared neighbor. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

projected_graph_adamic_adar

```
projected_graph_adamic_adar(nodes, neighbors=None, directed_method=None,
*, data=True)
```

Computes an Adamic-Adar projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than *nodes*.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'adamic_adar' edge attribute.

projected_graph_common_neighbors

```
projected_graph_common_neighbors(nodes, neighbors=None,
directed_method=None, *, data=True)
```

Computes a common neighbors projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'common_neighbors' edge attribute.

projected_graph_cosine

```
projected_graph_cosine(nodes, edge_weight=None, neighbors=None,
directed_method=None, *, data=True)
```

Computes a cosine projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'cosine' edge attribute.

projected_graph_dice

```
projected_graph_dice(nodes, neighbors=None, directed_method=None, *,
data=True)
```

Computes a Sørensen-Dice projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the subset of nodes that are specified by the *neighbors* parameter. The strength of association in the calculated projection is represented by the 'dice' edge attribute.

projected_graph_jaccard

```
projected_graph_jaccard(nodes, neighbors=None, directed_method=None, *,
data=True)
```

Computes a Jaccard projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'jaccard' edge attribute.

query

```
query(q, expr=None, _node_attr_to_ignore=None, _edge_attr_to_ignore=None, *,
induced=False, break_symmetry=False, _generate_output=True,
_orbits_only=False, **kwargs)
```

Queries for the subgraph isomorphisms of q within g.

Parameters

q : graph
Specifies the query graph.

expr : _Expression or None, optional
Specifies additional logic to filter the query results. The default is None.

induced : bool, optional
Specifies whether or not to return only node-induced matches. The default is False.

break_symmetry : bool, optional
Specifies whether or not to break symmetry and eliminate automorphic relationships between matches. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one isomorphic mapping from *q* to *self*. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

reach_graph

```
reach_graph(source, max_reach=1, *, data=True)
```

Computes the reach graph for the source or set of sources.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to compute the reach network.

max_reach : int, optional
Specifies the number of steps (or hops) to traverse from the source nodes. The default is 1.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns a graph that represents the reach network. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

reach_graph_per_source

```
reach_graph_per_source(source, max_reach=1, *, data=True)
```

Computes a separate reach graph for each given source node.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to compute the reach network.

max_reach : int, optional
Specifies the number of steps (or hops) to traverse from the source nodes. The default is 1.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one reach network. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

remove_edge

```
remove_edge(u, v)
```

Removes a single edge from the graph.

Parameters

u : numeric or str
Specifies the *from* node.

v : numeric or str
Specifies the *to* node.

Examples

```
G.remove_edge(1,2)
```

remove_edges_from

```
remove_edges_from(edges)
```

Removes specified edges from the graph.

Parameters

edges : an iterable container of edges
Specifies the edges to remove from the graph.

remove_isolated_nodes

```
remove_isolated_nodes(immutable=None, *, inplace=False)
```

Removes isolated nodes from a graph.

Parameters

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the input graph. The default is False.

Returns

SAS graph object or None
Returns the resulting graph after isolated nodes are removed, or returns nothing if the *inplace* parameter value is True.

remove_node

```
remove_node(node)
```

Removes a single node from the graph.

Parameters

node : numeric or str
Specifies the node to remove from the graph.

remove_nodes_from

```
remove_nodes_from(nodes)
```

Removes specified nodes from the graph.

If a specified node does not exist in the graph, it is silently ignored.

Parameters

nodes : an iterable container of nodes
Specifies the nodes to remove from the graph.

reverse

```
reverse(*, data=True)
```

Returns a copy of a directed graph with reversed edges.

Parameters

data : bool, optional
Specifies whether or not to retain attributes. The default is True, meaning that the node and edge attributes of the graph are retained.

Returns

directed graph object
Returns a directed graph that holds the reversed edges.

set_edge_attributes

```
set_edge_attributes(values, name=None, immutable=None, *, inplace=False)
```

Sets edge attributes by using a given value or dictionary of values.

Parameters

values : scalar or dict-like or Pandas DataFrame
Specifies what the edge attribute should be set to. When this parameter is set to a Pandas DataFrame, it is expected to have from, to, and edge_key (only for multigraphs) columns, and the other columns are treated as edge attributes to add. When *values* is a dictionary, the keys are edges and the values are either edge attribute values or dictionaries of attribute name-value pairs. For multigraphs, the edge tuples must be of the form (u, v, key). For simple graphs, the keys must be tuples of the form (u, v). When this parameter is not set to a dictionary, it is treated as a single attribute value that is then applied to every edge in *self*. The attribute name is specified by the *name* parameter.

name : str or None, optional
Specifies the name of the edge attribute to set if the *values* parameter is set to a scalar. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is False, a copy of the input graph with the new edge attributes is returned. If the *inplace* parameter value is True, the graph is modified by setting the specified edge attributes.

set_node_attributes

```
set_node_attributes(values, name=None, immutable=None, *, inplace=False)
```

Sets node attributes from a given value or dictionary of values.

Parameters

values : scalar or dict-like or Pandas DataFrame
Specifies what the node attribute should be set to. When *values* is not a dictionary, it is treated as a single attribute value that is then applied to every node in *self*. This means that if you provide a mutable object, such as a list, updates to that object are reflected in the node attribute for every node. The attribute name is specified by the *name* parameter. When *values* is a dictionary, the keys are nodes and the values are either node attribute values or dictionaries of attribute name-value pairs.

name : str or None, optional
Specifies the name of the node attribute to set if the *values* parameter is set to a scalar. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not the graph is modified in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by setting the specified node attributes. If the *inplace* parameter value is False, a copy of the input graph with the new node attributes is returned.

shortest_path_distances

```
shortest_path_distances(source=None, sink=None, edge_weight=None,
minedgewt=None, maxedgewt=None)
```

Calculates the shortest path distances in a graph.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) and restricts enumeration to the shortest paths that contain the specified source node(s). The default is None.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) and restricts enumeration to the shortest paths that contain the specified sink node(s). The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

Returns

dict
Returns a dictionary where each key is a (source, sink) tuple and its value is the corresponding shortest source-sink path distance.

similarity_adamic_adar

```
similarity_adamic_adar(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Adamic-Adar node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_common_neighbors

```
similarity_common_neighbors(source, sink=None, top_k=None, bottom_k=None,
*, exclude_source=False, exclude_existing_neighbors=False)
```

Computes common neighbors node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_cosine

```
similarity_cosine(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes cosine node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_dice

```
similarity_dice(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Sørensen-Dice node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_graph

```
similarity_graph(measure, min_score=2.220446049250313e-16, max_score=inf,
*, exclude_source=False, exclude_existing_neighbors=False, data=True,
**kwargs)
```

Computes a node similarity graph.

Parameters

measure : str, one of {'adamic_adar', 'common_neighbors', 'cosine', 'dice', 'jaccard'}
Specifies the metric to be used to compute the node similarity.

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the edge attribute that corresponds to the measure name. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_adamic_adar

```
similarity_graph_adamic_adar(min_score=2.220446049250313e-16,
max_score=inf, *, exclude_source=False, exclude_existing_neighbors=False,
data=True)
```

Computes an Adamic-Adar node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'adamic_adar' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_common_neighbors

```
similarity_graph_common_neighbors(min_score=2.220446049250313e-16,
max_score=inf, *, exclude_source=False, exclude_existing_neighbors=False,
data=True)
```

Computes a common neighbors node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'common_neighbors' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_cosine

```
similarity_graph_cosine(edge_weight=None,
min_score=2.220446049250313e-16, max_score=inf, *, exclude_source=False,
exclude_existing_neighbors=False, data=True)
```

Computes a cosine node similarity graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'cosine' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_dice

```
similarity_graph_dice(min_score=2.220446049250313e-16, max_score=inf, *,
exclude_source=False, exclude_existing_neighbors=False, data=True)
```

Computes a Sørensen-Dice node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the *dice* edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_jaccard

```
similarity_graph_jaccard(min_score=2.220446049250313e-16, max_score=inf, *,
exclude_source=False, exclude_existing_neighbors=False, data=True)
```

Computes a Jaccard node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'jaccard' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_jaccard

```
similarity_jaccard(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Jaccard node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

singleton_node_count

```
singleton_node_count()
```

Computes a graph's singleton node count.

Returns

int
Returns the number of singleton nodes.

successors

```
successors(n)
```

Returns an iterator over all successor nodes of n.

Parameters

n : node
Specifies a node in the graph.

Returns

iterator
Returns an iterator over successors of node n.

to_directed

```
to_directed()
```

Returns a directed copy of the input graph.

Returns

SAS graph object
Returns a directed copy of the input graph.

to_immutable

```
to_immutable()
```

Creates an immutable graph from the input graph.

Returns

SAS graph object
Returns an immutable SAS graph object.

to_mutable

```
to_mutable()
```

Creates a mutable graph from the input graph.

Returns

SAS graph object
Returns a mutable SAS graph object.

to_pandas_edges

```
to_pandas_edges(*, data=True, _for_conversion_to_immutable=False)
```

Returns the graph edges as a Pandas DataFrame.

Parameters

data : bool or list, optional
Specifies whether or not to retain attributes. The default is True. When the value is True (or when a list is given), the Pandas DataFrame includes all the edge attributes (or all the listed attributes).

Returns

dataframe
Returns a Pandas DataFrame that includes the graph edges.

to_pandas_nodes

```
to_pandas_nodes(*, data=True, _for_conversion_to_immutable=False)
```

Returns the graph nodes as a Pandas DataFrame.

Parameters

data : bool or list, optional
Specifies whether or not to retain attributes. The default is True. When the value is True (or when a list is given), the Pandas DataFrame includes all the node attributes (or all the listed attributes).

Returns

dataframe
Returns a Pandas DataFrame that includes the graph nodes.

to_undirected

```
to_undirected()
```

Returns a directed copy of the input graph.

Returns

SAS graph object
Returns a directed copy of the input graph.

topological_ordering

```
topological_ordering()
```

Finds a topological ordering of the graph's nodes.

Yields

generator
Yields a generator of nodes in topological order. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

transitive_closure

```
transitive_closure()
```

Calculates the transitive closure of a graph.

Returns

SAS graph object
Returns the transitive closure of the input graph that is returned as a SAS MultiGraph or MultiDiGraph object. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

triangle_count

```
triangle_count()
```

Computes a graph's triangle count.

Returns

int
Returns the number of triangles.

tsp_tour

```
tsp_tour(edge_weight=None, maxtime=None, minobjective=None, *,
inplace=False, use_milp=True, data=True, **options)
```

Calculates a traveling salesman problem (TSP) tour of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted TSP is calculated.

maxtime : float or None, optional
Specifies the maximum amount of time for solving a TSP. The default is None.

minobjective : float, optional
Specifies a minimum value such that when a solution is found whose objective is less than or equal to this value, the algorithm stops.

inplace : bool, optional
Specifies whether to add the TSP order values as attributes to the input graph directly or to create a copy of the graph. The default is False.

use_milp : bool, optional
Specifies whether or not to use a mixed integer linear programming (MILP) solver to solve a TSP. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is False, the returned graph is a Cycle that contains the calculated TSP tour edges. When the *inplace* parameter value is True, the graph is modified in place and None is returned. The modified graph contains the edge attributes *tsp* and *tsp_order* and the node attribute *tsp_order*.

union

```
union(rhs, nodes_attrs_agg=None, edges_attrs_agg=None)
```

Returns the union of input graphs by finding the unions of their nodes and edges.

The node attributes and edge attributes are aggregated using the rules that are defined by the *nodes_attrs_agg* and *edges_attrs_agg* parameters.

Parameters

rhs : SAS graph object
Specifies the other graph, of the same type as *self*, with which to find the union.

nodes_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate node attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps node attributes to functions. The default is None.

edges_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate edge attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps edge attributes to functions. The default is None.

Returns

SAS graph object
Returns a graph that is constructed by the union of the two graphs.

vrp

```
vrp(capacity, depot, node_demand='demand', edge_cost='weight', minroutes=1,
maxroutes=None, maxtime=None, minobjective=None, *, use_milp=True,
data=True, **options)
```

Solves the vehicle routing problem.

Parameters

capacity : float
Specifies the capacity of each vehicle.

depot : node
Specifies the depot node for the vehicle routing problem.

node_demand : str or dict or iterable
Specifies the quantity of goods to deliver to or pick up from a customer. The default is 'demand'.

edge_cost : str or dict or iterable
Specifies the cost of traversing each edge. The default is 'weight'.

minroutes : int, optional
Specifies the minimum number of routes that are allowed to service demand.

maxroutes : int, optional
Specifies the maximum number of routes that are allowed to service demand.

maxtime : float or None, optional
Specifies the maximum amount of time for solving the vehicle routing problem. The default is None.

minobjective : float, optional
Specifies a minimum value such that when a solution is found whose objective is less than or equal to this value, the algorithm stops.

use_milp : bool, optional
Specifies whether or not to use a mixed integer linear programming (MILP) solver to solve the vehicle routing problem. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Cycle objects; each cycle represents one route. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

vrptw

```
vrptw(capacity, depot, node_demand='demand', edge_cost='weight',
travel_time='weight', timewindow_lower=None, timewindow_upper=None,
service_time=None, minroutes=1, maxroutes=None, maxtime=None, *,
data=True)
```

Solves the vehicle routing problem with time windows.

Parameters

capacity : float
Specifies the capacity of each vehicle.

depot : node
Specifies the depot node for the vehicle routing problem.

node_demand : str or dict or iterable
Specifies the quantity of goods to deliver to or pick up from a customer. The default is 'demand'.

edge_cost : str or dict or iterable
Specifies the cost of traversing each edge.

travel_time : str or dict or iterable
Specifies attributes or values that define the time that it takes to traverse each edge. The default is 'weight'.

timewindow_lower : str or dict or iterable or None, optional
Specifies attributes or values that define the lower time limits within which a delivery vehicle can arrive at the customer nodes. The default is None.

timewindow_upper : str or dict or iterable or None, optional
Specifies attributes or values that define the upper time limits within which a delivery vehicle can arrive at the customer nodes. The default is None.

service_time : str or dict or iterable or None, optional
Specifies attributes or values that define the service time of the nodes. The default is None.

minroutes : int, optional
Specifies the minimum number of routes that are allowed to service demand.

maxroutes : int, optional
Specifies the maximum number of routes that are allowed to service demand.

maxtime : float or None, optional
Specifies the maximum amount of time for solving the vehicle routing problem. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Cycle objects; each cycle represents one route. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

Graph

Base class for undirected graphs.

API: Graph

class sasviya.network.classes.graph.Graph(*args, **kwargs)

Self-loops and multiedges are not allowed.

Attributes

edges
Returns the edges of the graph.

nodes
Returns the nodes of the graph.

Methods

Method Name	Description
add_core_number_attr	Computes the core numbers and stores the results as node attributes on the graph.
add_edge	Adds a single edge to the graph.
add_edges_from	Adds specified edges to the graph.
add_node	Adds a node to the graph.
add_nodes_from	Adds the specified nodes to the graph.
add_topological_order_attr	Computes a topological ordering and stores the results as node attributes of the graph.
add_weighted_edges_from	Adds weighted edges to the graph.

Method Name	Description
adjacency	Returns an iterator over (node, adjacency dict) for all nodes.
allows_multiedges	Specifies whether or not the graph allows multiedges.
allows_selfloops	Specifies whether or not the graph allows self-loops.
apply	Applies a function to all node or edge attributes and updates them.
approximate_diameter	Computes an approximation of the graph's diameter.
articulation_point_count	Computes a graph's articulation point count.
articulation_points	Finds the articulation points (cut points) of a graph.
assortativity_degree	Computes a graph's degree assortativity.
assortativity_nominal	Computes a graph's nominal assortativity.
assortativity_numeric	Computes a graph's numeric assortativity.
average_shortest_path	Computes a graph's average shortest path.
biconcomp_distribution_edges	Computes the distribution of the number of edges in each biconnected components by using *numpy.histogram*.
biconcomp_distribution_nodes	Computes the distribution of the number of nodes in each biconnected components by using *numpy.histogram*.
biconcomp_size_edges	Computes the number of edges within each biconnected component.
biconcomp_size_nodes	Computes the number of nodes within each biconnected component.
biconnected_component_count	Computes a graph's biconnected component count.
biconnected_components	Computes the graph's biconnected components for an undirected graph.
block_cut_tree	Computes the block-cut tree of a graph.
centrality_betweenness	Computes weighted or unweighted betweenness centrality and stores the results as attributes on the graph.

Method Name	Description
centrality_closeness	Computes weighted or unweighted closeness centrality and stores the results as attributes on the graph.
centrality_degree	Computes weighted or unweighted degree centrality and stores the results as attributes on the graph.
centrality_eigenvector	Computes weighted or unweighted eigenvector centrality and stores the results as attributes on the graph.
centrality_hub_authority	Computes weighted or unweighted hub and authority centrality and stores the results as attributes on the graph.
centrality_influence	Computes weighted or unweighted influence centrality and stores the results as attributes on the graph.
centrality_pagerank	Computes weighted or unweighted PageRank centrality and stores the results as attributes on the graph.
clear	Removes all nodes and edges from the graph.
clear_edges	Removes all edges from the graph but retains nodes.
concomp_distribution_edges	Computes the distribution of the number of edges in each connected component by using *numpy.histogram*.
concomp_distribution_nodes	Computes the distribution of the number of nodes in each connected component by using *numpy.histogram*.
concomp_size_edges	Computes the number of edges within each connected component.
concomp_size_nodes	Computes the number of nodes within each connected component.
connected_component_count	Computes a graph's connected component count.
connected_components	Computes the connected components of a graph.
copy	Returns a copy of the input graph.
core_decomposition	Computes a core decomposition of the graph.

Method Name	Description
cycle_count	Counts the number of cycles in a graph.
degree	Returns the degree of nodes.
density	Computes the density of the graph.
diameter	Computes a graph's diameter.
drop_edge_attributes	Removes edge attributes if they exist.
drop_node_attributes	Removes node attributes if they exist.
enumerate_cycle_edges	Enumerates the cycles in a graph and returns the edges of the cycles.
enumerate_cycle_nodes	Enumerates the cycles in a graph and returns the nodes of the cycles.
enumerate_cycles	Enumerates the cycles in a graph and returns the cycle graph objects.
enumerate_path_edges	Enumerates the paths in a graph and returns the edges of the paths.
enumerate_path_nodes	Enumerates the paths in a graph and returns the nodes of the paths.
enumerate_paths	Enumerates the paths in a graph and returns the Path graph objects.
enumerate_sequence_shortest_paths	Enumerates the shortest paths in a graph that visit the indicated sequence nodes in order.
enumerate_shortest_path_edges	Enumerates the shortest paths in a graph and returns the edges of the paths.
enumerate_shortest_path_nodes	Enumerates the shortest paths in a graph and returns the nodes of the paths.
enumerate_shortest_paths	Enumerates the shortest paths in a graph and returns the Path graph objects.
get_meta_graph	Constructs a metagraph from the input graph.
get_nlargest_biconcomps	Returns a list of up to *n* largest biconnected components.
get_nlargest_concomps	Returns a list of up to *n* largest connected (with highest number of nodes) components of a graph.

Method Name	Description
has_edge_attribute	Checks the edges of the input graph for the specified edge attribute.
has_node_attribute	Checks the nodes of the input graph for the specified node attribute.
intersection	Returns the intersection of input graphs by finding the intersections of their nodes and edges.
is_biconnected	Specifies whether or not the input graph is biconnected.
is_bipartite	Determines whether the input graph is bipartite.
is_connected	Specifies whether or not the input graph is connected.
is_cycle	Determines whether the input graph is a simple cycle.
is_dag	Determines whether the input graph is a directed acyclic graph (DAG).
is_directed	Specifies whether or not the graph is directed.
is_hamiltonian	Determines whether the graph is Hamiltonian (has a Hamiltonian cycle) or not.
is_immutable	Specifies whether or not the graph is immutable.
is_multigraph	Specifies whether or not the graph allows multiedges.
is_path	Determines whether the input graph is a simple path.
is_strongly_connected	Specifies whether the input graph is strongly connected.
is_weakly_connected	Specifies whether the input graph is weakly connected.
label_propagation	Computes the communities of the graph by using the label propagation algorithm.
label_propagation_nodes	Computes community assignments for all nodes in the graph by using the label propagation algorithm.
leaf_node_count	Computes a graph's leaf node count.

Method Name	Description
local_transitivity	Computes the local transitivity (clustering coefficient) and stores the results as node attributes on the graph.
louvain	Computes the communities of the graph by using the Louvain algorithm.
louvain_nodes	Computes community assignments for all nodes in the graph by using the Louvain algorithm.
maxflow	Calculates the maximum flow from a source node to a sink node in a graph.
maximal_cliques	Enumerates all maximal cliques of the graph.
min_cut	Calculates a minimum cut of a graph.
min_s_t_cut	Calculates a minimum s-t cut of a graph.
mincostflow	Calculates the minimum-cost network flow for the graph.
minimum_spanning_tree	Calculates a minimum spanning tree (MST) or forest of a graph.
minimum_weight_full_matching	Returns a minimum-weight full matching of the bipartite graph *self*.
neighbors	Returns an iterator over all neighbors of node n.
node_clique_numbers	Computes node clique numbers and stores the results as attributes on the graph.
number_of_edges	Returns the number of edges in the graph.
number_of_nodes	Returns the number of nodes in the graph.
projected_graph	Computes a projected graph.
projected_graph_adamic_adar	Computes an Adamic-Adar projected graph.
projected_graph_common_neighbors	Computes a common neighbors projected graph.
projected_graph_cosine	Computes a cosine projected graph.
projected_graph_dice	Computes a Sørensen-Dice projected graph.
projected_graph_jaccard	Computes a Jaccard projected graph.

Method Name	Description
query	Queries for the subgraph isomorphisms of q within g.
reach_graph	Computes the reach graph for the source or set of sources.
reach_graph_per_source	Computes a separate reach graph for each given source node.
remove_edge	Removes a single edge from the graph.
remove_edges_from	Removes specified edges from the graph.
remove_isolated_nodes	Removes isolated nodes from a graph.
remove_node	Removes a single node from the graph.
remove_nodes_from	Removes specified nodes from the graph.
set_edge_attributes	Sets edge attributes by using a given value or dictionary of values.
set_node_attributes	Sets node attributes from a given value or dictionary of values.
shortest_path_distances	Calculates the shortest path distances in a graph.
similarity_adamic_adar	Computes Adamic-Adar node similarity.
similarity_common_neighbors	Computes common neighbors node similarity.
similarity_cosine	Computes cosine node similarity.
similarity_dice	Computes Sørensen-Dice node similarity.
similarity_graph	Computes a node similarity graph.
similarity_graph_adamic_adar	Computes an Adamic-Adar node similarity graph.
similarity_graph_common_neighbors	Computes a common neighbors node similarity graph.
similarity_graph_cosine	Computes a cosine node similarity graph.
similarity_graph_dice	Computes a Sørensen-Dice node similarity graph.
similarity_graph_jaccard	Computes a Jaccard node similarity graph.
similarity_jaccard	Computes Jaccard node similarity.

Method Name	Description
singleton_node_count	Computes a graph's singleton node count.
to_directed	Returns a directed copy of the input graph.
to_immutable	Creates an immutable graph from the input graph.
to_mutable	Creates a mutable graph from the input graph.
to_pandas_edges	Returns the graph edges as a Pandas DataFrame.
to_pandas_nodes	Returns the graph nodes as a Pandas DataFrame.
to_undirected	Returns a directed copy of the input graph.
topological_ordering	Finds a topological ordering of the graph's nodes.
transitive_closure	Calculates the transitive closure of a graph.
triangle_count	Computes a graph's triangle count.
tsp_tour	Calculates a traveling salesman problem (TSP) tour of a graph.
union	Returns the union of input graphs by finding the unions of their nodes and edges.
vrp	Solves the vehicle routing problem.
vrptw	Solves the vehicle routing problem with time windows.

add_core_number_attr

```
add_core_number_attr(maxtime=None, *, inplace=False, data=True)
```

Computes the core numbers and stores the results as node attributes on the graph.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time for core decomposition to spend. The default is None.

inplace : bool, optional
Specifies whether to add the core numbers as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph with core numbers that are stored as a node attribute named *core*. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

add_edge

```
add_edge(u, v, error_on_selfloop=None, **attrs)
```

Adds a single edge to the graph.

Parameters

u : numeric or str
Specifies the *from* node.

v : numeric or str
Specifies the *to* node.

error_on_selfloop : bool or None, optional
Specifies whether or not to raise a ValueError if a self-loop is detected; that is, if (u==v). The default is None, which means that the input edge is silently ignored.

Examples

```
G.add_edge(1,2,weight=3,type='parent')
```

add_edges_from

```
add_edges_from(edges, error_on_selfloop=None, **kwargs)
```

Adds specified edges to the graph.

Parameters

edges : an iterable container of edges
Specifies the edges to add to the graph.

error_on_selfloop : bool or None, optional
Specifies whether or not to raise a ValueError if a self-loop is detected; that is, if (u==v). The default is None, which means that the input self-loop is silently ignored.

Examples

```
G.add_edges_from([("A","B",{"weight":1,"color":"blue"}),
                      ("C","D",{"weight":2,"color":"red"})])
G.add_edges_from([['X', 'Y'],["Y","Z"], weight=1,color= "blue")
```

add_node

```
add_node(node, **attrs)
```

Adds a node to the graph.

Parameters

node : str or number or tuple
Specifies the ID of the node to add to the graph. If the value is a tuple, the first element should contain the node ID, and the second element is a dictionary that represents the node's attributes.

add_nodes_from

```
add_nodes_from(nodes, **attrs)
```

Adds the specified nodes to the graph.

Parameters

nodes : an iterable container of nodes
Specifies the nodes to add to the graph.

Examples

```
G.add_nodes_from([('X', {"weight":11,"color": "blue"}), ("Y", {"color": "blue"})])
G.add_nodes_from(('X', 'Y'), weight=1,color= "blue")
G.add_nodes_from(['X', 'Y'])
```

add_topological_order_attr

add_topological_order_attr(*, inplace=False, data=True)

Computes a topological ordering and stores the results as node attributes of the graph.

Parameters

inplace : bool, optional
Specifies whether or not to add the topological order values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a directed graph whose topological order is stored as a node attribute named *top_order*. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

add_weighted_edges_from

add_weighted_edges_from(edges, weight='weight', **kwargs)

Adds weighted edges to the graph.

Parameters

edges : an iterable container of edges
Specifies the weighted edges to add to the graph.

weight : str
Specifies the attribute name for the weighted edges to add. The default is 'weight'.

adjacency

```
adjacency()
```

Returns an iterator over (node, adjacency dict) for all nodes.

For directed graphs, only outgoing adjacencies are returned.

Returns

iterator
Returns an iterator over (node, adjacency dictionary) for all nodes in the graph.

allows_multiedges

```
allows_multiedges()
```

Specifies whether or not the graph allows multiedges.

Returns

bool
Indicates whether or not the graph allows multiedges.

allows_selfloops

```
allows_selfloops()
```

Specifies whether or not the graph allows self-loops.

Returns

bool
Indicates whether or not the graph allows self-loops.

apply

apply(node_attr_handler=None, edge_attr_handler=None, immutable=None, *, inplace=False)

Applies a function to all node or edge attributes and updates them.

#use apply to fillna >>> sasnet.apply(G, edge_attr_handler=lambda data:{‘w’:0 if ‘w’ not in data else data[‘w’]},inplace=True) >>> assert G.edges[(1,2)][‘w’] == 0 >>> print(G.nodes(data=True)) #use apply to negate an attribute >>> sasnet.apply(G, node_attr_handler=lambda data:{‘x’: -data[‘x’]},inplace=True) >>> assert G.nodes[1] [‘x’]== -10

Parameters

node_attr_handler : function or None, optional
Specifies the function to apply to node attributes. The default is None.

edge_attr_handler : function or None, optional
Specifies the function to apply to edge attributes. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the input graph. The default is False.

Returns

SAS graph object or None.
Returns the resulting graph after the function is applied, or returns nothing if the *inplace* parameter value is True.

Examples

```
import sasviya.network as sasnet
G = sasnet.Graph([[1,2,{}],[1,3,{'w':1}],[1,4,{'w':2}]])
sasnet.set_node_attributes(G,{1:10,2:20,3:30,4:40},name='x',inplace=True)
```

#use apply to fillna >>> sasnet.apply(G, edge_attr_handler=lambda data:{‘w’:0 if ‘w’ not in data else data[‘w’]},inplace=True) >>> assert G.edges[(1,2)][‘w’] == 0 >>> print(G.nodes(data=True)) #use apply to negate an attribute >>> sasnet.apply(G, node_attr_handler=lambda data:{‘x’: -data[‘x’]},inplace=True) >>> assert G.nodes[1] [‘x’]== -10

approximate_diameter

```
approximate_diameter(edge_weight=None)
```

Computes an approximation of the graph's diameter.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted approximate diameter is computed.

Returns

float
Returns the approximate diameter.

articulation_point_count

```
articulation_point_count()
```

Computes a graph's articulation point count.

Returns

int
Returns the number of articulation points.

articulation_points

```
articulation_points()
```

Finds the articulation points (cut points) of a graph.

Returns

list
Returns a list of articulation points in the graph.

assortativity_degree

```
assortativity_degree(source_direction='out', target_direction='in',
edge_weight=None)
```

Computes a graph's degree assortativity.

Parameters

source_direction : {'in', 'out'}
Specifies the degree type (in-degree or out-degree) for the source node in a directed graph.

target_direction : {'in', 'out'}
Specifies the degree type (in-degree or out-degree) for the target node in a directed graph.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted assortativity degree is computed.

Returns

float
Returns the degree assortativity.

assortativity_nominal

```
assortativity_nominal(node_attr)
```

Computes a graph's nominal assortativity.

Parameters

node_attr : str
Specifies the name of the nominal node attribute to use.

Returns

float
Returns the nominal assortativity.

assortativity_numeric

```
assortativity_numeric(node_attr)
```

Computes a graph's numeric assortativity.

Parameters

node_attr : str
Specifies the name of the numeric node attribute to use.

Returns

float
Returns the numeric assortativity.

average_shortest_path

```
average_shortest_path(edge_weight=None)
```

Computes a graph's average shortest path.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted average shortest path is computed.

Returns

float
Returns the average shortest path.

biconcomp_distribution_edges

```
biconcomp_distribution_edges(bins=None)
```

Computes the distribution of the number of edges in each biconnected components by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is int (an integer), it specifies the number of equal-width bins. When *bins* is a list, it specifies bin ranges and should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is str (a string), it specifies the method to use for calculating the bin ranges. For example, 'stone','auto','doane','fd','rice','scott','sqrt','sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of biconnected components whose edge count is within the range of the bin.

biconcomp_distribution_nodes

```
biconcomp_distribution_nodes(bins=None)
```

Computes the distribution of the number of nodes in each biconnected components by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is int (an integer), it specifies the number of equal-width bins. When *bins* is a list, it specifies bin ranges and should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is str (a string), it specifies the method to use for calculating the bin ranges. For example, 'stone','auto','doane','fd','rice','scott','sqrt','sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of biconnected components whose node count is within the range of the bin.

biconcomp_size_edges

```
biconcomp_size_edges()
```

Computes the number of edges within each biconnected component.

Returns

dict
Returns a dictionary in which the keys are biconnected component IDs and the values are the number of edges within each component.

biconcomp_size_nodes

```
biconcomp_size_nodes()
```

Computes the number of nodes within each biconnected component.

Returns

dict
Returns a dictionary in which the keys are biconnected component IDs and the values are the number of nodes within each component.

biconnected_component_count

```
biconnected_component_count()
```

Computes a graph's biconnected component count.

Returns

int
Returns the number of biconnected components.

biconnected_components

```
biconnected_components(order_by=None, *, data=True, ascending=False)
```

Computes the graph's biconnected components for an undirected graph.

Parameters

order_by : 'node' or 'edge' or None, optional
Specifies the parameter that is used to sort the components by the number of nodes or edges within each component. The default is None, which means that the components are not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Returns the result as a generator of biconnected component subgraphs. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

block_cut_tree

```
block_cut_tree(*, return_nodes_map=False)
```

Computes the block-cut tree of a graph.

Parameters

return_nodes_map : bool, optional
Specifies whether or not to also compute the block-cut tree node map. The default is False.

Returns

SAS graph object
Returns a graph that represents the block-cut tree. When *return_nodes_map = True*, the output graph has the *nodes_map* attribute, which contains a dictionary that maps its nodes to a list of nodes in the original graph. Note: If a block is empty (that is, the corresponding biconnected component contains only articulation points), that block does not appear as a key in the *nodes_map* dictionary.

centrality_betweenness

```
centrality_betweenness(edge_weight=None, sample_percent=100, *,
inplace=False, data=True)
```

Computes weighted or unweighted betweenness centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that unweighted betweenness centrality is computed.

sample_percent : int, optional
Specifies the percentage of source nodes to sample for the approximate betweenness calculation. This parameter value should be in the range (0, 100]. The default is 100.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the node and edge attributes *between_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_closeness

```
centrality_closeness(edge_weight=None, nopath='diameter', *, inplace=False,
data=True)
```

Computes weighted or unweighted closeness centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted closeness centrality is computed.

nopath : {'diameter', 'harmonic', or 'zero'}, optional
Specifies a method to use for accounting for the shortest path distance between two nodes when a path does not exist. The valid values are as follows: - 'diameter' uses the graph diameter (plus one) as the shortest path distance between disconnected nodes. - 'harmonic' uses the harmonic formula for calculating closeness centrality. - 'zero' uses zero as the shortest path distance between disconnected nodes.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the closeness centrality node attributes *centr_close_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_degree

```
centrality_degree(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted degree centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted degree centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the degree centrality node attributes *centr_degree_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_eigenvector

```
centrality_eigenvector(edge_weight=None, maxiters=10000, *, inplace=False,
data=True)
```

Computes weighted or unweighted eigenvector centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted eigenvector centrality is computed.

maxiters : int, optional
Specifies the maximum number of iterations in order to limit the amount of computation time that is spent when convergence is slow. The default is 10,000.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the degree centrality node attributes *centr_eigen_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_hub_authority

```
centrality_hub_authority(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted hub and authority centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted hub and authority centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the hub and authority centrality node attributes *centr_hub_** and *centr_auth_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_influence

```
centrality_influence(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted influence centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted influence centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the first- and second-order influence centrality node attributes *centr_influence1_** and *centr_influence2_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_pagerank

```
centrality_pagerank(edge_weight=None, alpha=0.85, tolerance=1e-09, *,
inplace=False, data=True)
```

Computes weighted or unweighted PageRank centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted PageRank centrality is computed.

alpha : float, optional
Specifies the damping parameter for the PageRank centrality (or algorithm). The default is 0.85.

tolerance : float, optional
Specifies the tolerance parameter for the PageRank centrality (or algorithm). The default is 1E-9.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the PageRank centrality node attributes *centr_pagerank_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

clear

```
clear()
```

Removes all nodes and edges from the graph.

clear_edges

```
clear_edges()
```

Removes all edges from the graph but retains nodes.

concomp_distribution_edges

```
concomp_distribution_edges(bins=None, *, strongly=True)
```

Computes the distribution of the number of edges in each connected component by using *numpy.histogram*.

Parameters

bins : int or list or str, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is an integer (int), it represents the number of equal-width bins. When *bins* is a list, it specifies bin ranges and it should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)*

(inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is a string (str), it specifies the method to be used for calculating the bin ranges. For example, 'stone', 'auto', 'doane', 'fd', 'rice', 'scott', 'sqrt', 'sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of components whose edge count within the range of the bin.

concomp_distribution_nodes

```
concomp_distribution_nodes(bins=None, *, strongly=True)
```

Computes the distribution of the number of nodes in each connected component by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is an integer (int), it represents the number of equal-width bins. When *bins* is a list, it specifies bin ranges and it should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is a string (str), it specifies the method to use for calculating the bin ranges. For example, 'stone', 'auto', 'doane', 'fd', 'rice', 'scott', 'sqrt', 'sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of components whose node count is within the range of the bin.

concomp_size_edges

```
concomp_size_edges(*, strongly=True)
```

Computes the number of edges within each connected component.

Parameters

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary in which the keys are connected component IDs and the values are the number of edges within each component.

concomp_size_nodes

```
concomp_size_nodes(*, strongly=True)
```

Computes the number of nodes within each connected component.

Parameters

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary in which the keys are connected component IDs and the values are the number of nodes within each component.

connected_component_count

```
connected_component_count()
```

Computes a graph's connected component count.

Returns

int
Returns the number of connected components.

connected_components

```
connected_components(order_by=None, *, strongly=True, data=True,
ascending=False)
```

Computes the connected components of a graph.

Parameters

order_by : 'node' or 'edge' or None, optional
Specifies whether to sort the components by the number of nodes or edges within each component. The default is None, which means that the components are not sorted.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one connected component. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

copy

```
copy(immutable=None, *, data=True)
```

Returns a copy of the input graph.

Parameters

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, meaning that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns an independent copy of the input.

core_decomposition

```
core_decomposition(maxtime=None, *, data=True)
```

Computes a core decomposition of the graph.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time for core decomposition to spend. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one k-core in increasing core number. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

cycle_count

```
cycle_count(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Counts the number of cycles in a graph.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Returns

int
Returns a count of cycles in the graph.

degree

```
degree(n=None, edge_weight=None)
```

Returns the degree of nodes.

Parameters

n : single node or container of nodes or None, optional
Specifies which nodes to include. The default is None, which means that the degree of every node is returned.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None.

Returns

int or dictionary
Returns the degree of specified node(s).

density

```
density()
```

Computes the density of the graph.

Returns

float
Returns the density of the graph.

diameter

```
diameter(edge_weight=None)
```

Computes a graph's diameter.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted diameter is computed.

Returns

int or float
Returns the diameter.

drop_edge_attributes

```
drop_edge_attributes(attributes_subset=None, edges_subset=None,
immutable=None, *, inplace=False)
```

Removes edge attributes if they exist.

Parameters

attributes_subset : str or list, optional
Specifies the edge attribute or a list of the edge attributes to remove; for example, 'weight' or ['weight','color']. By default, all edge attributes are removed.

edges_subset : list or iterable of edges, optional
Specifies the edges to remove attributes from; for example, [(1,2),(2,3)]. By default the attributes for all edges are removed.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by removing the specified edge attributes. If the *inplace* parameter value is False, a copy of the input graph with the modified edge attributes is returned.

drop_node_attributes

```
drop_node_attributes(attributes_subset=None, nodes_subset=None,
immutable=None, *, inplace=False)
```

Removes node attributes if they exist.

Parameters

attributes_subset : str or list, optional
Specifies the node attribute or list of node attributes to remove; for example, ['weight','color']. By default, all node attributes are removed.

nodes_subset : list or iterable of nodes, optional
Specifies the nodes to drop attributes from; for example, [1,2,3] or (1,2,3) or [1] or (1,). By default, the attributes for all nodes are removed.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by removing the specified node attributes. If the *inplace* parameter value is False, a copy of the input graph with the modified node attributes is returned.

enumerate_cycle_edges

```
enumerate_cycle_edges(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Enumerates the cycles in a graph and returns the edges of the cycles.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Yields

generator
Yields a generator of cycles, each of which is given as a list of the cycles' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_cycle_nodes

```
enumerate_cycle_nodes(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Enumerates the cycles in a graph and returns the nodes of the cycles.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Yields

generator
Yields a generator of cycles, each of which is given as a list of the cycles' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_cycles

```
enumerate_cycles(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None, *,
data=True)
```

Enumerates the cycles in a graph and returns the cycle graph objects.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one cycle of the same type as the input graph. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_path_edges

```
enumerate_path_edges(source=None, sink=None, minlength=1,
maxlength=None, edge_weight=None, node_weight=None, minedgewt=None,
maxedgewt=None, minnodewt=None, maxnodewt=None, maxtime=None)
```

Enumerates the paths in a graph and returns the edges of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

Yields

generator
Yields a generator of Path objects, each of which is given as a list of the paths' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_path_nodes

```
enumerate_path_nodes(source=None, sink=None, minlength=1,
maxlength=None, edge_weight=None, node_weight=None, minedgewt=None,
maxedgewt=None, minnodewt=None, maxnodewt=None, maxtime=None)
```

Enumerates the paths in a graph and returns the nodes of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

Yields

generator
Yields a generator of Path objects, each of which is given as a list of the paths' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_paths

```
enumerate_paths(source=None, sink=None, minlength=1, maxlength=None,
edge_weight=None, node_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, *, data=True)
```

Enumerates the paths in a graph and returns the Path graph objects.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one Path object. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_sequence_shortest_paths

```
enumerate_sequence_shortest_paths(sequence, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None, *, data=True)
```

Enumerates the shortest paths in a graph that visit the indicated sequence nodes in order.

Parameters

sequence : iterable of nodes
Specifies the order in which to visit the required nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Path objects; each object represents a subpath between a sequence of nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_path_edges

```
enumerate_shortest_path_edges(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None)
```

Enumerates the shortest paths in a graph and returns the edges of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

Yields

generator
Yields a generator of paths; each path is given as a list of the paths' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_path_nodes

```
enumerate_shortest_path_nodes(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None)
```

Enumerates the shortest paths in a graph and returns the nodes of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

Yields

generator
Yields a generator of paths; each path is given as a list of the paths' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_paths

```
enumerate_shortest_paths(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None, *, data=True)
```

Enumerates the shortest paths in a graph and returns the Path graph objects.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Path objects; each object represents one shortest path. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

get_meta_graph

```
get_meta_graph(node_type, edge_type=None, immutable=None)
```

Constructs a metagraph from the input graph.

Parameters

node_type : str
Specifies the node attribute to consider in constructing the metagraph.

edge_type : str, optional
Specifies the edge attribute to consider in constructing the metagraph. By default, all connections between nodes are considered the same.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

Returns

SAS graph object
Returns the metagraph, which has an edge attribute count and determines the number of unique connections between pairs of nodes.

get_nlargest_biconcomps

```
get_nlargest_biconcomps(n, order_by='node', *, data=True, ascending=False)
```

Returns a list of up to *n* largest biconnected components.

Parameters

n : int
Specifies the number of graphs to return.

order_by : 'node' or 'edge' or None
Specifies whether to sort the biconnected components by the number of nodes or edges within those components. The default is None, which means that the biconnected components are not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. When the value is True, the smallest biconnected components are returned instead. The default is False.

Returns

list
Returns a list of n largest or smallest biconnected components of a graph.

get_nlargest_concomps

```
get_nlargest_concomps(n, order_by='node', *, strongly=True, data=True,
ascending=False)
```

Returns a list of up to *n* largest connected (with highest number of nodes) components of a graph.

Parameters

n : int
Specifies the number of graphs to return.

order_by : 'node' or 'edge'
Specifies whether to sort the component by number of nodes or edges within that component. The default is None, which means that the result is not sorted.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Returns

list
Returns a list of the n largest connected components of a graph.

has_edge_attribute

```
has_edge_attribute(name, *, all_edges=False)
```

Checks the edges of the input graph for the specified edge attribute.

Parameters

name : str
Specifies the name of the edge attribute.

all_edges : bool, optional
Specifies whether or not to check for the existence of the edge attribute in all or any of the edges. The default is False.

Returns

bool
If the all_edges parameter value is True, the value True is returned if that edge attribute exists in all edges; otherwise the value False is returned. If the all_edges parameter value is False, the value True is returned if at least one edge in the graph has that specified edge attribute.

has_node_attribute

```
has_node_attribute(name, *, all_nodes=False)
```

Checks the nodes of the input graph for the specified node attribute.

Parameters

name : str
Specifies the name of the node attribute.

all_nodes : bool, optional
Specifies whether or not to check for the existence of the node attribute in all or any of the nodes. The default is False.

Returns

bool
If the all_nodes parameter value is True, the value True is returned if that node attribute exists in all node; otherwise the value False is returned. If the all_nodes parameter value is False, the value True is returned if at least one node in the graph has that specified node attribute.

intersection

```
intersection(rhs, nodes_attrs_agg=None, edges_attrs_agg=None)
```

Returns the intersection of input graphs by finding the intersections of their nodes and edges.

The node attributes and edge attributes are aggregated using the rules that are defined by the *nodes_attrs_agg* and *edges_attrs_agg* parameters.

Parameters

rhs : SAS graph object
Specifies the other graph, of the same type as *self*, with which to intersect.

nodes_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate node attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps node attributes to functions. The default is None.

edges_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate edge attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps edge attributes to functions. The default is None.

Returns

SAS graph object
Returns a graph that is constructed by the intersection of the two graphs.

is_biconnected

```
is_biconnected()
```

Specifies whether or not the input graph is biconnected.

Returns

bool
Indicates whether the input graph is biconnected (True) or not (False).

is_bipartite

```
is_bipartite()
```

Determines whether the input graph is bipartite.

Returns

bool
Indicates whether the input graph is bipartite.

is_connected

```
is_connected()
```

Specifies whether or not the input graph is connected.

Returns

bool
Indicates whether the input graph is connected (True) or not (False).

is_cycle

```
is_cycle()
```

Determines whether the input graph is a simple cycle.

Returns

bool
Indicates whether the input graph is a simple cycle (True) or not (False).

is_dag

```
is_dag()
```

Determines whether the input graph is a directed acyclic graph (DAG).

Returns

bool
Indicates whether the input graph is a directed acyclic graph (DAG).

is_directed

```
is_directed()
```

Specifies whether or not the graph is directed.

Returns

bool
Indicates whether or not the graph is directed.

is_hamiltonian

```
is_hamiltonian(maxtime=None)
```

Determines whether the graph is Hamiltonian (has a Hamiltonian cycle) or not.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time to determine whether or not the graph is Hamiltonian. The default is None.

Returns

bool or None
Indicates whether or not the input graph is determined to be Hamiltonian. Returns None if no determination is made.

is_immutable

```
is_immutable()
```

Specifies whether or not the graph is immutable.

Returns

bool
Indicates whether or not the graph is immutable.

is_multigraph

```
is_multigraph()
```

Specifies whether or not the graph allows multiedges.

Returns

bool
Indicates whether or not the graph allows multiedges.

is_path

```
is_path()
```

Determines whether the input graph is a simple path.

Returns

bool
Indicates whether the input graph is a simple path (True) or not (False).

is_strongly_connected

```
is_strongly_connected()
```

Specifies whether the input graph is strongly connected.

Returns

bool
Indicates whether the input graph is strongly connected (True) or not (False).

is_weakly_connected

```
is_weakly_connected()
```

Specifies whether the input graph is weakly connected.

Returns

bool
Indicates whether the input graph is weakly connected (True) or not (False).

label_propagation

```
label_propagation(edge_weight=None, maxiters=100, tolerance=0.05, *,
data=True)
```

Computes the communities of the graph by using the label propagation algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of label changes for all nodes in the graph is less than the value. The default is 0.05.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one community. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

label_propagation_nodes

```
label_propagation_nodes(edge_weight=None, maxiters=100, tolerance=0.05, *,
data=True, inplace=False)
```

Computes community assignments for all nodes in the graph by using the label propagation algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of label changes for all nodes in the graph is less than the value. The default is 0.05.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

inplace : bool, optional
Specifies whether to add the community values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains the node attribute *community*, which denotes the community assignment for each node. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

leaf_node_count

```
leaf_node_count()
```

Computes a graph's leaf node count.

Returns

int
Returns the number of leaf nodes.

local_transitivity

```
local_transitivity(*, inplace=False)
```

Computes the local transitivity (clustering coefficient) and stores the results as node attributes on the graph.

Parameters

inplace : bool, optional
Specifies whether to add the transitivity values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains node attribute 'transitivity'. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

louvain

```
louvain(*, edge_weight=None, resolution=1, tolerance=0.001, maxiters=100,
data=True)
```

Computes the communities of the graph by using the Louvain algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

resolution : float, optional
Specifies the factor in the modularity equation that affects the expected number of links between two nodes. A higher value produces more communities. The default is 1.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of modularity gain between two consecutive iterations is less than the value. The default is 0.001.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one community. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

louvain_nodes

```
louvain_nodes(*, edge_weight=None, resolution=1, tolerance=0.001,
maxiters=100, data=True, inplace=False)
```

Computes community assignments for all nodes in the graph by using the Louvain algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

resolution : float, optional
Specifies the factor in the modularity equation that affects the expected number of links between two nodes. A higher value produces more communities. The default is 1.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of modularity gain between two consecutive iterations is less than the value. The default is 0.001.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

inplace : bool, optional
Specifies whether to add the community values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains the node attribute *community*, which denotes the community assignment for each node. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

maxflow

```
maxflow(source, sink, capacity_attr='upper', *, inplace=False, data=True)
```

Calculates the maximum flow from a source node to a sink node in a graph.

Parameters

source : number or str
Specifies the source node for the maximum flow.

sink : number or str
Specifies the sink node for the maximum flow.

capacity_attr : str, optional
Specifies the name of the edge attribute that contains the capacity of the edge. The default is 'upper'.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute *flow*, which contains the flow values. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, the returned graph contains only the flow edges.

maximal_cliques

```
maximal_cliques(maxcliques='ALL', node_weight=None, edge_weight=None,
minsize=1, maxsize=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, order_by=None, *,
data=True, ascending=False)
```

Enumerates all maximal cliques of the graph.

Parameters

maxcliques : int or 'ALL', optional
Specifies the maximum number of enumerated cliques to return. The default is 'ALL'.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

minsize : int, optional
Specifies the minimum number of nodes in a clique. The default is 1.

maxsize : int or None, optional
Specifies the maximum number of nodes in a clique. The default is None.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a clique. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a clique. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a clique. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a clique. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds for enumeration to spend. The default is None.

order_by : 'node' or None, optional
Specifies whether to sort the cliques by the number of nodes within each clique. The default is None, which means that the result is not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one maximal clique. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

min_cut

```
min_cut(edge_weight=None, *, inplace=False, data=True)
```

Calculates a minimum cut of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted minimum cut is calculated.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes

of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute 'mincut', which indicates the edges to include in the minimum cut set, and the node attribute 'mincut_partition', which indicates the node partition of the calculated minimum cut. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, a modified copy of the graph is returned.

min_s_t_cut

```
min_s_t_cut(source, sink, edge_weight=None, *, inplace=False, data=True)
```

Calculates a minimum s-t cut of a graph.

Parameters

source : number or str
Specifies the source node (s) for the minimum s-t cut.

sink : number or str
Specifies the sink node (t) for the minimum s-t cut.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted minimum s-t cut is calculated.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute *s_t_cut*, which indicates the edges to include in the minimum s-t cut set; and the node attribute *s_t_cut_partition*, which indicates the node partition of the calculated minimum s-t cut. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is

modified in place and None is returned. When the *inplace* parameter value is False, a modified copy of the graph is returned.

mincostflow

```
mincostflow(demand_lower_attr='lower', demand_upper_attr='upper',
capacity_lower_attr='lower', capacity_upper_attr='upper', cost_attr='weight',
maxtime=None, *, inplace=False, data=True)
```

Calculates the minimum-cost network flow for the graph.

Parameters

demand_lower_attr : str, optional
Specifies the name of the node attribute that contains the lower bound of the node supply. The default is 'lower'.

demand_upper_attr : str, optional
Specifies the name of the node attribute that contains the upper bound of the node supply. The default is 'upper'.

capacity_lower_attr : str, optional
Specifies the name of the edge attribute that contains the capacity lower bound of the edge. The default is 'lower'.

capacity_upper_attr : str, optional
Specifies the name of the edge attribute that contains the capacity upper bound of the edge. The default is 'upper'.

cost_attr : str, optional
Specifies the name of the edge attribute that contains the edge cost. The default is 'weight'.

maxtime : float or None, optional
Specifies the maximum amount of time to allow for computing the minimum-cost network flow. The default is None.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attributes *flow* and *reduced_cost*, which contain the flow and reduced cost values, respectively; and the node attribute *dual*, which contains the optimal dual value. This graph has the

solution_summary attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned.

minimum_spanning_tree

```
minimum_spanning_tree(edge_weight=None, source=None, *, inplace=False,
data=True)
```

Calculates a minimum spanning tree (MST) or forest of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

source : number or str or None
Specifies the source node for a directed MST. The default is None.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute 'mst', which indicates the edges to include in the spanning tree or forest. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, the returned graph contains only the spanning tree or forest edges.

minimum_weight_full_matching

```
minimum_weight_full_matching(edge_weight=None, *, data=True)
```

Returns a minimum-weight full matching of the bipartite graph *self*.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns a directed graph whose edges identify the assignment mapping in the minimum-weight full matching solution. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

neighbors

```
neighbors(n)
```

Returns an iterator over all neighbors of node n.

Parameters

n : node
Specifies a node in the graph.

Returns

iterator
Returns an iterator over all neighbors of node n.

node_clique_numbers

```
node_clique_numbers(*, inplace=False, data=True)
```

Computes node clique numbers and stores the results as attributes on the graph.

Parameters

inplace : bool, optional
Specifies whether to add the clique number values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the value of the *inplace* parameter is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph or modifies the existing graph to contain the node attribute *clique_number*, which includes the clique number of each node. The graph g has the attribute g.solution_summary, which contains information about the results of the algorithm.

number_of_edges

```
number_of_edges(u=None, v=None)
```

Returns the number of edges in the graph.

When you specify *u* and *v*, the number of edges between *u* and *v* is returned. When you specify *u* but *v* is None, the number of edges that connect to *u* is returned. When you specify *v* but *u* is None, the number of edges that connect to *v* is returned.

Parameters

u : numeric or str or None, optional
Specifies the *from* node. The default is None.

v : numeric or str or None, optional
Specifies the *to* node. The default is None.

Returns

int
Returns the number of edges in the graph. When you specify *u* and *v*, the number of edges between *u* and *v* is returned. When you specify *u* but *v* is None, the number of edges that connect to *u* is returned. When you specify *v* but *u* is None, the number of edges that connect to *v* is returned.

number_of_nodes

```
number_of_nodes()
```

Returns the number of nodes in the graph.

Returns

int
Returns the number of nodes in the graph.

projected_graph

```
projected_graph(nodes, neighbors=None, directed_method=None,
_measure=None, *, multigraph=False, data=True)
```

Computes a projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in self other than *nodes*.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used. When the *directed_method* parameter is specified for an undirected graph, a ValueError is raised.

multigraph : bool, optional
Specifies whether or not to return a multigraph. When the value is True, the algorithm returns a multigraph in which the multiple edges represent multiple shared neighbors. The edge attribute 'neighbor' stores the label of the shared neighbor. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

projected_graph_adamic_adar

```
projected_graph_adamic_adar(nodes, neighbors=None, directed_method=None,
*, data=True)
```

Computes an Adamic-Adar projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than *nodes*.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'adamic_adar' edge attribute.

projected_graph_common_neighbors

```
projected_graph_common_neighbors(nodes, neighbors=None,
directed_method=None, *, data=True)
```

Computes a common neighbors projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'common_neighbors' edge attribute.

projected_graph_cosine

```
projected_graph_cosine(nodes, edge_weight=None, neighbors=None,
directed_method=None, *, data=True)
```

Computes a cosine projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'cosine' edge attribute.

projected_graph_dice

```
projected_graph_dice(nodes, neighbors=None, directed_method=None, *,
data=True)
```

Computes a Sørensen-Dice projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the subset of nodes that are specified by the *neighbors* parameter. The strength of association in the calculated projection is represented by the 'dice' edge attribute.

projected_graph_jaccard

```
projected_graph_jaccard(nodes, neighbors=None, directed_method=None, *,
data=True)
```

Computes a Jaccard projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'jaccard' edge attribute.

query

```
query(q, expr=None, _node_attr_to_ignore=None, _edge_attr_to_ignore=None, *,
induced=False, break_symmetry=False, _generate_output=True,
_orbits_only=False, **kwargs)
```

Queries for the subgraph isomorphisms of q within g.

Parameters

q : graph
Specifies the query graph.

expr : _Expression or None, optional
Specifies additional logic to filter the query results. The default is None.

induced : bool, optional
Specifies whether or not to return only node-induced matches. The default is False.

break_symmetry : bool, optional
Specifies whether or not to break symmetry and eliminate automorphic relationships between matches. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one isomorphic mapping from *q* to *self*. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

reach_graph

```
reach_graph(source, max_reach=1, *, data=True)
```

Computes the reach graph for the source or set of sources.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to compute the reach network.

max_reach : int, optional
Specifies the number of steps (or hops) to traverse from the source nodes. The default is 1.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns a graph that represents the reach network. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

reach_graph_per_source

```
reach_graph_per_source(source, max_reach=1, *, data=True)
```

Computes a separate reach graph for each given source node.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to compute the reach network.

max_reach : int, optional
Specifies the number of steps (or hops) to traverse from the source nodes. The default is 1.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one reach network. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

remove_edge

```
remove_edge(u, v)
```

Removes a single edge from the graph.

Parameters

u : numeric or str
Specifies the *from* node.

v : numeric or str
Specifies the *to* node.

Examples

```
G.remove_edge(1,2)
```

remove_edges_from

```
remove_edges_from(edges)
```

Removes specified edges from the graph.

Parameters

edges : an iterable container of edges
Specifies the edges to remove from the graph.

remove_isolated_nodes

```
remove_isolated_nodes(immutable=None, *, inplace=False)
```

Removes isolated nodes from a graph.

Parameters

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the input graph. The default is False.

Returns

SAS graph object or None
Returns the resulting graph after isolated nodes are removed, or returns nothing if the *inplace* parameter value is True.

remove_node

```
remove_node(node)
```

Removes a single node from the graph.

Parameters

node : Numeric/String
Specifies the node to remove from the graph.

remove_nodes_from

```
remove_nodes_from(nodes)
```

Removes specified nodes from the graph.

If a specified node does not exist in the graph, it is silently ignored.

Parameters

nodes : an iterable container of nodes
Specifies the nodes to remove from the graph.

set_edge_attributes

```
set_edge_attributes(values, name=None, immutable=None, *, inplace=False)
```

Sets edge attributes by using a given value or dictionary of values.

Parameters

values : scalar or dict-like or Pandas DataFrame
Specifies what the edge attribute should be set to. When this parameter is set to a Pandas DataFrame, it is expected to have from, to, and edge_key (only for multigraphs) columns, and the other columns are treated as edge attributes to add. When *values* is a dictionary, the keys are edges and the values are either edge attribute values or dictionaries of attribute name-value pairs. For multigraphs, the edge tuples must be of the form (u, v, key). For simple graphs, the keys must be tuples of the form (u, v). When this parameter is not set to a dictionary, it is treated as a single attribute value that is then applied to every edge in *self*. The attribute name is specified by the *name* parameter.

name : str or None, optional
Specifies the name of the edge attribute to set if the *values* parameter is set to a scalar. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is False, a copy of the input graph with the new edge attributes is returned. If the *inplace* parameter value is True, the graph is modified by setting the specified edge attributes.

set_node_attributes

```
set_node_attributes(values, name=None, immutable=None, *, inplace=False)
```

Sets node attributes from a given value or dictionary of values.

Parameters

values : scalar or dict-like or Pandas DataFrame
Specifies what the node attribute should be set to. When *values* is not a dictionary, it is treated as a single attribute value that is then applied to every node in *self*. This means that if you provide a mutable object, such as a list, updates to that object are reflected in the node attribute for every node. The attribute name is specified by the *name* parameter. When *values* is a dictionary, the keys are nodes and the values are either node attribute values or dictionaries of attribute name-value pairs.

name : str or None, optional
Specifies the name of the node attribute to set if the *values* parameter is set to a scalar. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not the graph is modified in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by setting the specified node attributes. If the *inplace* parameter value is False, a copy of the input graph with the new node attributes is returned.

shortest_path_distances

```
shortest_path_distances(source=None, sink=None, edge_weight=None,
minedgewt=None, maxedgewt=None)
```

Calculates the shortest path distances in a graph.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) and restricts enumeration to the shortest paths that contain the specified source node(s). The default is None.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) and restricts enumeration to the shortest paths that contain the specified sink node(s). The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

Returns

dict
Returns a dictionary where each key is a (source, sink) tuple and its value is the corresponding shortest source-sink path distance.

similarity_adamic_adar

```
similarity_adamic_adar(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Adamic-Adar node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_common_neighbors

```
similarity_common_neighbors(source, sink=None, top_k=None, bottom_k=None, *, exclude_source=False, exclude_existing_neighbors=False)
```

Computes common neighbors node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_cosine

```
similarity_cosine(source, sink=None, top_k=None, bottom_k=None, *, exclude_source=False, exclude_existing_neighbors=False)
```

Computes cosine node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_dice

```
similarity_dice(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Sørensen-Dice node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_graph

```
similarity_graph(measure, min_score=2.220446049250313e-16, max_score=inf,
*, exclude_source=False, exclude_existing_neighbors=False, data=True,
**kwargs)
```

Computes a node similarity graph.

Parameters

measure : str, one of {'adamic_adar', 'common_neighbors', 'cosine', 'dice', 'jaccard'}
Specifies the metric to be used to compute the node similarity.

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the edge attribute that corresponds to the measure name. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_adamic_adar

```
similarity_graph_adamic_adar(min_score=2.220446049250313e-16,
max_score=inf, *, exclude_source=False, exclude_existing_neighbors=False,
data=True)
```

Computes an Adamic-Adar node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'adamic_adar' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_common_neighbors

```
similarity_graph_common_neighbors(min_score=2.220446049250313e-16,
max_score=inf, *, exclude_source=False, exclude_existing_neighbors=False,
data=True)
```

Computes a common neighbors node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'common_neighbors' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_cosine

```
similarity_graph_cosine(edge_weight=None,
min_score=2.220446049250313e-16, max_score=inf, *, exclude_source=False,
exclude_existing_neighbors=False, data=True)
```

Computes a cosine node similarity graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'cosine' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_dice

```
similarity_graph_dice(min_score=2.220446049250313e-16, max_score=inf, *,
exclude_source=False, exclude_existing_neighbors=False, data=True)
```

Computes a Sørensen-Dice node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the *dice* edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_jaccard

```
similarity_graph_jaccard(min_score=2.220446049250313e-16, max_score=inf, *,
exclude_source=False, exclude_existing_neighbors=False, data=True)
```

Computes a Jaccard node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'jaccard' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_jaccard

```
similarity_jaccard(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Jaccard node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

singleton_node_count

```
singleton_node_count()
```

Computes a graph's singleton node count.

Returns

int
Returns the number of singleton nodes.

to_directed

```
to_directed()
```

Returns a directed copy of the input graph.

Returns

SAS graph object
Returns a directed copy of the input graph.

to_immutable

```
to_immutable()
```

Creates an immutable graph from the input graph.

Returns

SAS graph object
Returns an immutable SAS graph object.

to_mutable

```
to_mutable()
```

Creates a mutable graph from the input graph.

Returns

SAS graph object
Returns a mutable SAS graph object.

to_pandas_edges

```
to_pandas_edges(*, data=True, _for_conversion_to_immutable=False)
```

Returns the graph edges as a Pandas DataFrame.

Parameters

data : bool or list, optional
Specifies whether or not to retain attributes. The default is True. When the value is True (or when a list is given), the Pandas DataFrame includes all the edge attributes (or all the listed attributes).

Returns

dataframe
Returns a Pandas DataFrame that includes the graph edges.

to_pandas_nodes

```
to_pandas_nodes(*, data=True, _for_conversion_to_immutable=False)
```

Returns the graph nodes as a Pandas DataFrame.

Parameters

data : bool or list, optional
Specifies whether or not to retain attributes. The default is True. When the value is True (or when a list is given), the Pandas DataFrame includes all the node attributes (or all the listed attributes).

Returns

dataframe
Returns a Pandas DataFrame that includes the graph nodes.

to_undirected

```
to_undirected()
```

Returns a directed copy of the input graph.

Returns

SAS graph object
Returns a directed copy of the input graph.

topological_ordering

```
topological_ordering()
```

Finds a topological ordering of the graph's nodes.

Yields

generator
Yields a generator of nodes in topological order. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

transitive_closure

```
transitive_closure()
```

Calculates the transitive closure of a graph.

Returns

SAS graph object
Returns the transitive closure of the input graph that is returned as a SAS MultiGraph or MultiDiGraph object. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

triangle_count

```
triangle_count()
```

Computes a graph's triangle count.

Returns

int
Returns the number of triangles.

tsp_tour

```
tsp_tour(edge_weight=None, maxtime=None, minobjective=None, *,
inplace=False, use_milp=True, data=True, **options)
```

Calculates a traveling salesman problem (TSP) tour of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted TSP is calculated.

maxtime : float or None, optional
Specifies the maximum amount of time for solving a TSP. The default is None.

minobjective : float, optional
Specifies a minimum value such that when a solution is found whose objective is less than or equal to this value, the algorithm stops.

inplace : bool, optional
Specifies whether to add the TSP order values as attributes to the input graph directly or to create a copy of the graph. The default is False.

use_milp : bool, optional
Specifies whether or not to use a mixed integer linear programming (MILP) solver to solve a TSP. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is False, the returned graph is a Cycle that contains the calculated TSP tour edges. When the *inplace* parameter value is True, the graph is modified in place and None is returned. The modified graph contains the edge attributes *tsp* and *tsp_order* and the node attribute *tsp_order*.

union

```
union(rhs, nodes_attrs_agg=None, edges_attrs_agg=None)
```

Returns the union of input graphs by finding the unions of their nodes and edges.

The node attributes and edge attributes are aggregated using the rules that are defined by the *nodes_attrs_agg* and *edges_attrs_agg* parameters.

Parameters

rhs : SAS graph object
Specifies the other graph, of the same type as *self*, with which to find the union.

nodes_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate node attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps node attributes to functions. The default is None.

edges_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate edge attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps edge attributes to functions. The default is None.

Returns

SAS graph object
Returns a graph that is constructed by the union of the two graphs.

vrp

```
vrp(capacity, depot, node_demand='demand', edge_cost='weight', minroutes=1,
maxroutes=None, maxtime=None, minobjective=None, *, use_milp=True,
data=True, **options)
```

Solves the vehicle routing problem.

Parameters

capacity : float
Specifies the capacity of each vehicle.

depot : node
Specifies the depot node for the vehicle routing problem.

node_demand : str or dict or iterable
Specifies the quantity of goods to deliver to or pick up from a customer. The default is 'demand'.

edge_cost : str or dict or iterable
Specifies the cost of traversing each edge. The default is 'weight'.

minroutes : int, optional
Specifies the minimum number of routes that are allowed to service demand.

maxroutes : int, optional
Specifies the maximum number of routes that are allowed to service demand.

maxtime : float or None, optional
Specifies the maximum amount of time for solving the vehicle routing problem. The default is None.

minobjective : float, optional
Specifies a minimum value such that when a solution is found whose objective is less than or equal to this value, the algorithm stops.

use_milp : bool, optional
Specifies whether or not to use a mixed integer linear programming (MILP) solver to solve the vehicle routing problem. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Cycle objects; each cycle represents one route. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

vrptw

```
vrptw(capacity, depot, node_demand='demand', edge_cost='weight',
travel_time='weight', timewindow_lower=None, timewindow_upper=None,
service_time=None, minroutes=1, maxroutes=None, maxtime=None, *,
data=True)
```

Solves the vehicle routing problem with time windows.

Parameters

capacity : float
Specifies the capacity of each vehicle.

depot : node
Specifies the depot node for the vehicle routing problem.

node_demand : str or dict or iterable
Specifies the quantity of goods to deliver to or pick up from a customer. The default is 'demand'.

edge_cost : str or dict or iterable
Specifies the cost of traversing each edge.

travel_time : str or dict or iterable
Specifies attributes or values that define the time that it takes to traverse each edge. The default is 'weight'.

timewindow_lower : str or dict or iterable or None, optional
Specifies attributes or values that define the lower time limits within which a delivery vehicle can arrive at the customer nodes. The default is None.

timewindow_upper : str or dict or iterable or None, optional
Specifies attributes or values that define the upper time limits within which a delivery vehicle can arrive at the customer nodes. The default is None.

service_time : str or dict or iterable or None, optional
Specifies attributes or values that define the service time of the nodes. The default is None.

minroutes : int, optional
Specifies the minimum number of routes that are allowed to service demand.

maxroutes : int, optional
Specifies the maximum number of routes that are allowed to service demand.

maxtime : float or None, optional
Specifies the maximum amount of time for solving the vehicle routing problem. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Cycle objects; each cycle represents one route. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

MultiDiGraph

Base class for directed multigraphs.

API: MultiDiGraph

class sasviya.network.classes.multidigraph.MultiDiGraph(*args, **kwargs)

Self-loops and multiedges are allowed.

Attributes

edges
Returns the edges of the graph.

nodes
Returns the nodes of the graph.

Methods

Method Name	Description
add_core_number_attr	Computes the core numbers and stores the results as node attributes on the graph.
add_edge	Adds a single edge to the graph.
add_edges_from	Adds edges to the graph.
add_node	Adds a node to the graph.

Method Name	Description
add_nodes_from	Adds the specified nodes to the graph.
add_topological_order_attr	Computes a topological ordering and stores the results as node attributes of the graph.
add_weighted_edges_from	Adds weighted edges to the graph.
adjacency	Returns an iterator over (node, adjacency dict) for all nodes.
allows_multiedges	Specifies whether or not the graph allows multiedges.
allows_selfloops	Specifies whether or not the graph allows self-loops.
apply	Applies a function to all node or edge attributes and updates them.
approximate_diameter	Computes an approximation of the graph's diameter.
articulation_point_count	Computes a graph's articulation point count.
articulation_points	Finds the articulation points (cut points) of a graph.
assortativity_degree	Computes a graph's degree assortativity.
assortativity_nominal	Computes a graph's nominal assortativity.
assortativity_numeric	Computes a graph's numeric assortativity.
average_shortest_path	Computes a graph's average shortest path.
biconcomp_distribution_edges	Computes the distribution of the number of edges in each biconnected components by using *numpy.histogram*.
biconcomp_distribution_nodes	Computes the distribution of the number of nodes in each biconnected components by using *numpy.histogram*.
biconcomp_size_edges	Computes the number of edges within each biconnected component.
biconcomp_size_nodes	Computes the number of nodes within each biconnected component.
biconnected_component_count	Computes a graph's biconnected component count.
biconnected_components	Computes the graph's biconnected components for an undirected graph.

Method Name	Description
block_cut_tree	Computes the block-cut tree of a graph.
centrality_betweenness	Computes weighted or unweighted betweenness centrality and stores the results as attributes on the graph.
centrality_closeness	Computes weighted or unweighted closeness centrality and stores the results as attributes on the graph.
centrality_degree	Computes weighted or unweighted degree centrality and stores the results as attributes on the graph.
centrality_eigenvector	Computes weighted or unweighted eigenvector centrality and stores the results as attributes on the graph.
centrality_hub_authority	Computes weighted or unweighted hub and authority centrality and stores the results as attributes on the graph.
centrality_influence	Computes weighted or unweighted influence centrality and stores the results as attributes on the graph.
centrality_pagerank	Computes weighted or unweighted PageRank centrality and stores the results as attributes on the graph.
clear	Removes all nodes and edges from the graph.
clear_edges	Removes all edges from the graph but retains nodes.
concomp_distribution_edges	Computes the distribution of the number of edges in each connected component by using *numpy.histogram*.
concomp_distribution_nodes	Computes the distribution of the number of nodes in each connected component by using *numpy.histogram*.
concomp_size_edges	Computes the number of edges within each connected component.
concomp_size_nodes	Computes the number of nodes within each connected component.
connected_component_count	Computes a graph's connected component count.

Method Name	Description
connected_components	Computes the connected components of a graph.
copy	Returns a copy of the input graph.
core_decomposition	Computes a core decomposition of the graph.
cycle_count	Counts the number of cycles in a graph.
degree	Returns the degree of nodes.
density	Computes the density of the graph.
diameter	Computes a graph's diameter.
drop_edge_attributes	Removes edge attributes if they exist.
drop_node_attributes	Removes node attributes if they exist.
enumerate_cycle_edges	Enumerates the cycles in a graph and returns the edges of the cycles.
enumerate_cycle_nodes	Enumerates the cycles in a graph and returns the nodes of the cycles.
enumerate_cycles	Enumerates the cycles in a graph and returns the cycle graph objects.
enumerate_path_edges	Enumerates the paths in a graph and returns the edges of the paths.
enumerate_path_nodes	Enumerates the paths in a graph and returns the nodes of the paths.
enumerate_paths	Enumerates the paths in a graph and returns the Path graph objects.
enumerate_sequence_shortest_paths	Enumerates the shortest paths in a graph that visit the indicated sequence nodes in order.
enumerate_shortest_path_edges	Enumerates the shortest paths in a graph and returns the edges of the paths.
enumerate_shortest_path_nodes	Enumerates the shortest paths in a graph and returns the nodes of the paths.
enumerate_shortest_paths	Enumerates the shortest paths in a graph and returns the Path graph objects.
get_meta_graph	Constructs a metagraph from the input graph.

Method Name	Description
get_nlargest_biconcomps	Returns a list of up to *n* largest biconnected components.
get_nlargest_concomps	Returns a list of up to *n* largest connected (with highest number of nodes) components of a graph.
has_edge_attribute	Checks the edges of the input graph for the specified edge attribute.
has_node_attribute	Checks the nodes of the input graph for the specified node attribute.
in_degree	Returns the in-degree of nodes.
intersection	Returns the intersection of input graphs by finding the intersections of their nodes and edges.
is_biconnected	Specifies whether or not the input graph is biconnected.
is_bipartite	Determines whether the input graph is bipartite.
is_connected	Specifies whether or not the input graph is connected.
is_cycle	Determines whether the input graph is a simple cycle.
is_dag	Determines whether the input graph is a directed acyclic graph (DAG).
is_directed	Specifies whether or not the graph is directed.
is_hamiltonian	Determines whether the graph is Hamiltonian (has a Hamiltonian cycle) or not.
is_immutable	Specifies whether or not the graph is immutable.
is_multigraph	Specifies whether or not the graph allows multiedges.
is_path	Determines whether the input graph is a simple path.
is_strongly_connected	Specifies whether the input graph is strongly connected.
is_weakly_connected	Specifies whether the input graph is weakly connected.
label_propagation	Computes the communities of the graph by using the label propagation algorithm.

Method Name	Description
label_propagation_nodes	Computes community assignments for all nodes in the graph by using the label propagation algorithm.
leaf_node_count	Computes a graph's leaf node count.
local_transitivity	Computes the local transitivity (clustering coefficient) and stores the results as node attributes on the graph.
louvain	Computes the communities of the graph by using the Louvain algorithm.
louvain_nodes	Computes community assignments for all nodes in the graph by using the Louvain algorithm.
maxflow	Calculates the maximum flow from a source node to a sink node in a graph.
maximal_cliques	Enumerates all maximal cliques of the graph.
min_cut	Calculates a minimum cut of a graph.
min_s_t_cut	Calculates a minimum s-t cut of a graph.
mincostflow	Calculates the minimum-cost network flow for the graph.
minimum_spanning_tree	Calculates a minimum spanning tree (MST) or forest of a graph.
minimum_weight_full_matching	Returns a minimum-weight full matching of the bipartite graph *self*.
neighbors	Returns an iterator over all successor nodes of n.
new_edge_key	Returns an unused key for the edges between nodes *u* and *v*.
node_clique_numbers	Computes node clique numbers and stores the results as attributes on the graph.
number_of_edges	Returns the number of edges in the graph.
number_of_nodes	Returns the number of nodes in the graph.
out_degree	Returns the out-degree of nodes.
predecessors	Returns an iterator over all predecessor nodes of n.
projected_graph	Computes a projected graph.

Method Name	Description
projected_graph_adamic_adar	Computes an Adamic-Adar projected graph.
projected_graph_common_neighbors	Computes a common neighbors projected graph.
projected_graph_cosine	Computes a cosine projected graph.
projected_graph_dice	Computes a Sørensen-Dice projected graph.
projected_graph_jaccard	Computes a Jaccard projected graph.
query	Queries for the subgraph isomorphisms of q within g.
reach_graph	Computes the reach graph for the source or set of sources.
reach_graph_per_source	Computes a separate reach graph for each given source node.
remove_edge	Removes a single edge from the graph.
remove_edges_from	Removes the edges from the graph.
remove_isolated_nodes	Removes isolated nodes from a graph.
remove_node	Removes a single node from the graph.
remove_nodes_from	Removes nodes from the graph. If a node does not exist in the graph, it is silently ignored.
reverse	Returns a copy of a directed graph with reversed edges.
set_edge_attributes	Sets edge attributes by using a given value or dictionary of values.
set_node_attributes	Sets node attributes from a given value or dictionary of values.
shortest_path_distances	Calculates the shortest path distances in a graph.
similarity_adamic_adar	Computes Adamic-Adar node similarity.
similarity_common_neighbors	Computes common neighbors node similarity.
similarity_cosine	Computes cosine node similarity.
similarity_dice	Computes Sørensen-Dice node similarity.

Method Name	Description
similarity_graph	Computes a node similarity graph.
similarity_graph_adamic_adar	Computes an Adamic-Adar node similarity graph.
similarity_graph_common_neighbors	Computes a common neighbors node similarity graph.
similarity_graph_cosine	Computes a cosine node similarity graph.
similarity_graph_dice	Computes a Sørensen-Dice node similarity graph.
similarity_graph_jaccard	Computes a Jaccard node similarity graph.
similarity_jaccard	Computes Jaccard node similarity.
singleton_node_count	Computes a graph's singleton node count.
successors	Returns an iterator over all successor nodes of n.
to_directed	Returns a directed copy of the input graph.
to_immutable	Creates an immutable graph from the input graph.
to_mutable	Creates a mutable graph from the input graph.
to_pandas_edges	Returns the graph edges as a Pandas DataFrame.
to_pandas_nodes	Returns the graph nodes as a Pandas DataFrame.
to_undirected	Returns a directed copy of the input graph.
topological_ordering	Finds a topological ordering of the graph's nodes.
transitive_closure	Calculates the transitive closure of a graph.
triangle_count	Computes a graph's triangle count.
tsp_tour	Calculates a traveling salesman problem (TSP) tour of a graph.
union	Returns the union of input graphs by finding the unions of their nodes and edges.
vrp	Solves the vehicle routing problem.
vrptw	Solves the vehicle routing problem with time windows.

add_core_number_attr

```
add_core_number_attr(maxtime=None, *, inplace=False, data=True)
```

Computes the core numbers and stores the results as node attributes on the graph.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time for core decomposition to spend. The default is None.

inplace : bool, optional
Specifies whether to add the core numbers as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph with core numbers that are stored as a node attribute named *core*. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

add_edge

```
add_edge(u, v, key=None, **kwargs)
```

Adds a single edge to the graph.

Parameters

u : numeric or str
Specifies the *from* node.

v : numeric or str
Specifies the *to* node.

key : numeric or str
Specifies the key of the edge to add to the graph. The default is None.

Returns

numeric or str
Returns the key of the edge that is added to the graph.

Examples

```
G.add_edge(1,2,key='A',weight=3,type='parent')
```

add_edges_from

```
add_edges_from(edges, **kwargs)
```

Adds edges to the graph.

Parameters

edges : an iterable container of edges
Specifies the edges to add to the graph.

Returns

list
Returns the keys of the edges that are added to the graph.

Examples

```
G.add_edges_from([("A","B",0,{"weight":1,"color":"blue"}),
                  ("C","D",0,{"weight":2,"color":"red"})])
G.add_edges_from([["X","Y",0],["Y","Z",0], weight=1,color= "blue")
```

add_node

```
add_node(node, **attrs)
```

Adds a node to the graph.

Parameters

node : str or number or tuple
Specifies the ID of the node to add to the graph. If the value is a tuple, the first element should contain the node ID, and the second element is a dictionary that represents the node's attributes.

add_nodes_from

```
add_nodes_from(nodes, **attrs)
```

Adds the specified nodes to the graph.

Parameters

nodes : an iterable container of nodes
Specifies the nodes to add to the graph.

Examples

```
G.add_nodes_from([('X', {"weight":11,"color": "blue"}), ("Y", {"color": "blue"})])
G.add_nodes_from(('X', 'Y'), weight=1,color= "blue")
G.add_nodes_from(['X', 'Y'])
```

add_topological_order_attr

```
add_topological_order_attr(*, inplace=False, data=True)
```

Computes a topological ordering and stores the results as node attributes of the graph.

Parameters

inplace : bool, optional
Specifies whether or not to add the topological order values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a directed graph whose topological order is stored as a node attribute named *top_order*. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

add_weighted_edges_from

```
add_weighted_edges_from(edges, weight='weight', **kwargs)
```

Adds weighted edges to the graph.

Parameters

edges : an iterable container of edges
Specifies the weighted edges to add to the graph.

weight : str
Specifies the attribute name for the weighted edges to add. The default is 'weight'.

adjacency

```
adjacency()
```

Returns an iterator over (node, adjacency dict) for all nodes.

For directed graphs, only outgoing adjacencies are returned.

Returns

iterator
Returns an iterator over (node, adjacency dictionary) for all nodes in the graph.

allows_multiedges

```
allows_multiedges()
```

Specifies whether or not the graph allows multiedges.

Returns

bool
Indicates whether or not the graph allows multiedges.

allows_selfloops

```
allows_selfloops()
```

Specifies whether or not the graph allows self-loops.

Returns

bool
Indicates whether or not the graph allows self-loops.

apply

```
apply(node_attr_handler=None, edge_attr_handler=None, immutable=None, *,
inplace=False)
```

Applies a function to all node or edge attributes and updates them.

#use apply to fillna >>> sasnet.apply(G, edge_attr_handler=lambda data:{'w':0 if 'w' not in data else data['w']},inplace=True) >>> assert G.edges[(1,2)]['w'] == 0 >>> print(G.nodes(data=True)) #use apply to negate an attribute >>> sasnet.apply(G, node_attr_handler=lambda data:{'x': -data['x']},inplace=True) >>> assert G.nodes[1]['x']== -10

Parameters

node_attr_handler : function or None, optional
Specifies the function to apply to node attributes. The default is None.

edge_attr_handler : function or None, optional
Specifies the function to apply to edge attributes. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the input graph. The default is False.

Returns

SAS graph object or None.
Returns the resulting graph after the function is applied, or returns nothing if the *inplace* parameter value is True.

Examples

```
import sasviya.network as sasnet
G = sasnet.Graph([[1,2,{}],[1,3,{'w':1}],[1,4,{'w':2}]])
sasnet.set_node_attributes(G,{1:10,2:20,3:30,4:40},name='x',inplace=True)
```

#use apply to fillna >>> sasnet.apply(G, edge_attr_handler=lambda data:{‘w’:0 if ‘w’ not in data else data[‘w’]},inplace=True) >>> assert G.edges[(1,2)][‘w’] == 0 >>> print(G.nodes(data=True)) #use apply to negate an attribute >>> sasnet.apply(G, node_attr_handler=lambda data:{‘x’: -data[‘x’]},inplace=True) >>> assert G.nodes[1][‘x’]== -10

approximate_diameter

```
approximate_diameter(edge_weight=None)
```

Computes an approximation of the graph’s diameter.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted approximate diameter is computed.

Returns

float
Returns the approximate diameter.

articulation_point_count

```
articulation_point_count()
```

Computes a graph’s articulation point count.

Returns

int
Returns the number of articulation points.

articulation_points

```
articulation_points()
```

Finds the articulation points (cut points) of a graph.

Returns

list
Returns a list of articulation points in the graph.

assortativity_degree

```
assortativity_degree(source_direction='out', target_direction='in',
edge_weight=None)
```

Computes a graph's degree assortativity.

Parameters

source_direction : {'in', 'out'}
Specifies the degree type (in-degree or out-degree) for the source node in a directed graph.

target_direction : {'in', 'out'}
Specifies the degree type (in-degree or out-degree) for the target node in a directed graph.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted assortativity degree is computed.

Returns

float
Returns the degree assortativity.

assortativity_nominal

```
assortativity_nominal(node_attr)
```

Computes a graph's nominal assortativity.

Parameters

node_attr : str
Specifies the name of the nominal node attribute to use.

Returns

float
Returns the nominal assortativity.

assortativity_numeric

```
assortativity_numeric(node_attr)
```

Computes a graph's numeric assortativity.

Parameters

node_attr : str
Specifies the name of the numeric node attribute to use.

Returns

float
Returns the numeric assortativity.

average_shortest_path

```
average_shortest_path(edge_weight=None)
```

Computes a graph's average shortest path.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted average shortest path is computed.

Returns

float
Returns the average shortest path.

biconcomp_distribution_edges

```
biconcomp_distribution_edges(bins=None)
```

Computes the distribution of the number of edges in each biconnected components by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is int (an integer), it specifies the number of equal-width bins. When *bins* is a list, it specifies bin ranges and should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is str (a string), it specifies the method to use for calculating the bin ranges. For example, 'stone','auto','doane','fd','rice','scott','sqrt','sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of biconnected components whose edge count is within the range of the bin.

biconcomp_distribution_nodes

```
biconcomp_distribution_nodes(bins=None)
```

Computes the distribution of the number of nodes in each biconnected components by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is int (an integer), it specifies the number of equal-width bins. When *bins* is a list, it specifies bin ranges and should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is str (a string), it specifies the method to use for calculating the bin ranges. For example, 'stone','auto','doane','fd','rice','scott','sqrt','sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of biconnected components whose node count is within the range of the bin.

biconcomp_size_edges

```
biconcomp_size_edges()
```

Computes the number of edges within each biconnected component.

Returns

dict
Returns a dictionary in which the keys are biconnected component IDs and the values are the number of edges within each component.

biconcomp_size_nodes

```
biconcomp_size_nodes()
```

Computes the number of nodes within each biconnected component.

Returns

dict
Returns a dictionary in which the keys are biconnected component IDs and the values are the number of nodes within each component.

biconnected_component_count

```
biconnected_component_count()
```

Computes a graph's biconnected component count.

Returns

int
Returns the number of biconnected components.

biconnected_components

```
biconnected_components(order_by=None, *, data=True, ascending=False)
```

Computes the graph's biconnected components for an undirected graph.

Parameters

order_by : 'node' or 'edge' or None, optional
Specifies the parameter that is used to sort the components by the number of nodes or edges within each component. The default is None, which means that the components are not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Returns the result as a generator of biconnected component subgraphs. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

block_cut_tree

```
block_cut_tree(*, return_nodes_map=False)
```

Computes the block-cut tree of a graph.

Parameters

return_nodes_map : bool, optional
Specifies whether or not to also compute the block-cut tree node map. The default is False.

Returns

SAS graph object
Returns a graph that represents the block-cut tree. When *return_nodes_map = True*, the output graph has the *nodes_map* attribute, which contains a dictionary that maps its nodes to a list of nodes in the original graph. Note: If a block is empty (that is, the corresponding biconnected component contains only articulation points), that block does not appear as a key in the *nodes_map* dictionary.

centrality_betweenness

```
centrality_betweenness(edge_weight=None, sample_percent=100, *,
inplace=False, data=True)
```

Computes weighted or unweighted betweenness centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that unweighted betweenness centrality is computed.

sample_percent : int, optional
Specifies the percentage of source nodes to sample for the approximate betweenness calculation. This parameter value should be in the range (0, 100]. The default is 100.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the node and edge attributes *between_*. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_closeness

```
centrality_closeness(edge_weight=None, nopath='diameter', *, inplace=False,
data=True)
```

Computes weighted or unweighted closeness centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted closeness centrality is computed.

nopath : {'diameter', 'harmonic', or 'zero'}, optional
Specifies a method to use for accounting for the shortest path distance between two nodes when a path does not exist. The valid values are as follows: - 'diameter' uses the graph diameter (plus one) as the shortest path distance between disconnected nodes. - 'harmonic' uses the harmonic formula for calculating closeness centrality. - 'zero' uses zero as the shortest path distance between disconnected nodes.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the closeness centrality node attributes *centr_close_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_degree

```
centrality_degree(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted degree centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted degree centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the degree centrality node attributes *centr_degree_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_eigenvector

```
centrality_eigenvector(edge_weight=None, maxiters=10000, *, inplace=False,
data=True)
```

Computes weighted or unweighted eigenvector centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted eigenvector centrality is computed.

maxiters : int, optional
Specifies the maximum number of iterations in order to limit the amount of computation time that is spent when convergence is slow. The default is 10,000.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the degree centrality node attributes *centr_eigen_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_hub_authority

```
centrality_hub_authority(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted hub and authority centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted hub and authority centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the hub and authority centrality node attributes *centr_hub_** and *centr_auth_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_influence

```
centrality_influence(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted influence centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted influence centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the first- and second-order influence centrality node attributes *centr_influence1_** and *centr_influence2_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_pagerank

```
centrality_pagerank(edge_weight=None, alpha=0.85, tolerance=1e-09, *,
inplace=False, data=True)
```

Computes weighted or unweighted PageRank centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted PageRank centrality is computed.

alpha : float, optional
Specifies the damping parameter for the PageRank centrality (or algorithm). The default is 0.85.

tolerance : float, optional
Specifies the tolerance parameter for the PageRank centrality (or algorithm). The default is 1E-9.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the PageRank centrality node attributes *centr_pagerank_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

clear

```
clear()
```

Removes all nodes and edges from the graph.

clear_edges

```
clear_edges()
```

Removes all edges from the graph but retains nodes.

concomp_distribution_edges

```
concomp_distribution_edges(bins=None, *, strongly=True)
```

Computes the distribution of the number of edges in each connected component by using *numpy.histogram*.

Parameters

bins : int or list or str, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is an integer (int), it represents the number of equal-width bins. When *bins* is a list, it specifies bin ranges and it should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is a string (str), it specifies the method to be used for calculating the bin ranges. For example, 'stone', 'auto', 'doane', 'fd', 'rice', 'scott', 'sqrt', 'sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of components whose edge count within the range of the bin.

concomp_distribution_nodes

```
concomp_distribution_nodes(bins=None, *, strongly=True)
```

Computes the distribution of the number of nodes in each connected component by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is an integer (int), it represents the number of equal-width bins. When *bins* is a list, it specifies bin ranges and it should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is a string (str), it specifies the method to use for calculating the bin ranges. For example, 'stone', 'auto', 'doane', 'fd', 'rice', 'scott', 'sqrt', 'sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of components whose node count is within the range of the bin.

concomp_size_edges

```
concomp_size_edges(*, strongly=True)
```

Computes the number of edges within each connected component.

Parameters

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary in which the keys are connected component IDs and the values are the number of edges within each component.

concomp_size_nodes

```
concomp_size_nodes(*, strongly=True)
```

Computes the number of nodes within each connected component.

Parameters

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary in which the keys are connected component IDs and the values are the number of nodes within each component.

connected_component_count

```
connected_component_count()
```

Computes a graph's connected component count.

Returns

int
Returns the number of connected components.

connected_components

```
connected_components(order_by=None, *, strongly=True, data=True,
ascending=False)
```

Computes the connected components of a graph.

Parameters

order_by : 'node' or 'edge' or None, optional
Specifies whether to sort the components by the number of nodes or edges within each component. The default is None, which means that the components are not sorted.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one connected component. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

copy

```
copy(immutable=None, *, data=True)
```

Returns a copy of the input graph.

Parameters

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, meaning that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns an independent copy of the input.

core_decomposition

```
core_decomposition(maxtime=None, *, data=True)
```

Computes a core decomposition of the graph.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time for core decomposition to spend. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one k-core in increasing core number. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

cycle_count

```
cycle_count(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Counts the number of cycles in a graph.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Returns

int
Returns a count of cycles in the graph.

degree

```
degree(n=None, edge_weight=None)
```

Returns the degree of nodes.

Parameters

n : single node or container of nodes or None, optional
Specifies which nodes to include. The default is None, which means that the degree of every node is returned.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None.

Returns

int or dictionary
Returns the degree of specified node(s).

density

```
density()
```

Computes the density of the graph.

Returns

float
Returns the density of the graph.

diameter

```
diameter(edge_weight=None)
```

Computes a graph's diameter.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted diameter is computed.

Returns

int or float
Returns the diameter.

drop_edge_attributes

```
drop_edge_attributes(attributes_subset=None, edges_subset=None,
immutable=None, *, inplace=False)
```

Removes edge attributes if they exist.

Parameters

attributes_subset : str or list, optional
Specifies the edge attribute or a list of the edge attributes to remove; for example, 'weight' or ['weight','color']. By default, all edge attributes are removed.

edges_subset : list or iterable of edges, optional
Specifies the edges to remove attributes from; for example, [(1,2),(2,3)]. By default the attributes for all edges are removed.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by removing the specified edge attributes. If the *inplace* parameter value is False, a copy of the input graph with the modified edge attributes is returned.

drop_node_attributes

```
drop_node_attributes(attributes_subset=None, nodes_subset=None,
immutable=None, *, inplace=False)
```

Removes node attributes if they exist.

Parameters

attributes_subset : str or list, optional
Specifies the node attribute or list of node attributes to remove; for example, ['weight','color']. By default, all node attributes are removed.

nodes_subset : list or iterable of nodes, optional
Specifies the nodes to drop attributes from; for example, [1,2,3] or (1,2,3) or [1] or (1,). By default, the attributes for all nodes are removed.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by removing the specified node attributes. If the *inplace* parameter value is False, a copy of the input graph with the modified node attributes is returned.

enumerate_cycle_edges

```
enumerate_cycle_edges(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Enumerates the cycles in a graph and returns the edges of the cycles.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Yields

generator
Yields a generator of cycles, each of which is given as a list of the cycles' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_cycle_nodes

```
enumerate_cycle_nodes(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Enumerates the cycles in a graph and returns the nodes of the cycles.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Yields

generator
Yields a generator of cycles, each of which is given as a list of the cycles' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_cycles

```
enumerate_cycles(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None, *,
data=True)
```

Enumerates the cycles in a graph and returns the cycle graph objects.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one cycle of the same type as the input graph. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_path_edges

```
enumerate_path_edges(source=None, sink=None, minlength=1,
maxlength=None, edge_weight=None, node_weight=None, minedgewt=None,
maxedgewt=None, minnodewt=None, maxnodewt=None, maxtime=None)
```

Enumerates the paths in a graph and returns the edges of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

Yields

generator
Yields a generator of Path objects, each of which is given as a list of the paths' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_path_nodes

```
enumerate_path_nodes(source=None, sink=None, minlength=1,
maxlength=None, edge_weight=None, node_weight=None, minedgewt=None,
maxedgewt=None, minnodewt=None, maxnodewt=None, maxtime=None)
```

Enumerates the paths in a graph and returns the nodes of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

Yields

generator
Yields a generator of Path objects, each of which is given as a list of the paths' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_paths

```
enumerate_paths(source=None, sink=None, minlength=1, maxlength=None,
edge_weight=None, node_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, *, data=True)
```

Enumerates the paths in a graph and returns the Path graph objects.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one Path object. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_sequence_shortest_paths

```
enumerate_sequence_shortest_paths(sequence, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None, *, data=True)
```

Enumerates the shortest paths in a graph that visit the indicated sequence nodes in order.

Parameters

sequence : iterable of nodes
Specifies the order in which to visit the required nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Path objects; each object represents a subpath between a sequence of nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_path_edges

```
enumerate_shortest_path_edges(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None)
```

Enumerates the shortest paths in a graph and returns the edges of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

Yields

generator
Yields a generator of paths; each path is given as a list of the paths' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_path_nodes

```
enumerate_shortest_path_nodes(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None)
```

Enumerates the shortest paths in a graph and returns the nodes of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

Yields

generator
Yields a generator of paths; each path is given as a list of the paths' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_paths

```
enumerate_shortest_paths(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None, *, data=True)
```

Enumerates the shortest paths in a graph and returns the Path graph objects.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Path objects; each object represents one shortest path. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

get_meta_graph

```
get_meta_graph(node_type, edge_type=None, immutable=None)
```

Constructs a metagraph from the input graph.

Parameters

node_type : str
Specifies the node attribute to consider in constructing the metagraph.

edge_type : str, optional
Specifies the edge attribute to consider in constructing the metagraph. By default, all connections between nodes are considered the same.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

Returns

SAS graph object
Returns the metagraph, which has an edge attribute count and determines the number of unique connections between pairs of nodes.

get_nlargest_biconcomps

```
get_nlargest_biconcomps(n, order_by='node', *, data=True, ascending=False)
```

Returns a list of up to *n* largest biconnected components.

Parameters

n : int
Specifies the number of graphs to return.

order_by : 'node' or 'edge' or None
Specifies whether to sort the biconnected components by the number of nodes or edges within those components. The default is None, which means that the biconnected components are not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. When the value is True, the smallest biconnected components are returned instead. The default is False.

Returns

list
Returns a list of n largest or smallest biconnected components of a graph.

get_nlargest_concomps

```
get_nlargest_concomps(n, order_by='node', *, strongly=True, data=True,
ascending=False)
```

Returns a list of up to *n* largest connected (with highest number of nodes) components of a graph.

Parameters

n : int
Specifies the number of graphs to return.

order_by : 'node' or 'edge'
Specifies whether to sort the component by number of nodes or edges within that component. The default is None, which means that the result is not sorted.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Returns

list
Returns a list of the n largest connected components of a graph.

has_edge_attribute

```
has_edge_attribute(name, *, all_edges=False)
```

Checks the edges of the input graph for the specified edge attribute.

Parameters

name : str
Specifies the name of the edge attribute.

all_edges : bool, optional
Specifies whether or not to check for the existence of the edge attribute in all or any of the edges. The default is False.

Returns

bool
If the all_edges parameter value is True, the value True is returned if that edge attribute exists in all edges; otherwise the value False is returned. If the all_edges parameter value is False, the value True is returned if at least one edge in the graph has that specified edge attribute.

has_node_attribute

```
has_node_attribute(name, *, all_nodes=False)
```

Checks the nodes of the input graph for the specified node attribute.

Parameters

name : str
Specifies the name of the node attribute.

all_nodes : bool, optional
Specifies whether or not to check for the existence of the node attribute in all or any of the nodes. The default is False.

Returns

bool
If the all_nodes parameter value is True, the value True is returned if that node attribute exists in all node; otherwise the value False is returned. If the

all_nodes parameter value is False, the value True is returned if at least one node in the graph has that specified node attribute.

in_degree

```
in_degree(n=None, edge_weight=None)
```

Returns the in-degree of nodes.

Parameters

n : single node or container of nodes or None, optional
Specifies which nodes to include. The default is None, which means that the in-degree of all nodes is returned.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None.

Returns

numeric or dictionary
Returns the in-degree of the node if a single node is passed, or returns a dictionary with nodes as keys and their in-degrees as values if None or a list of nodes is passed.

intersection

```
intersection(rhs, nodes_attrs_agg=None, edges_attrs_agg=None)
```

Returns the intersection of input graphs by finding the intersections of their nodes and edges.

The node attributes and edge attributes are aggregated using the rules that are defined by the *nodes_attrs_agg* and *edges_attrs_agg* parameters.

Parameters

rhs : SAS graph object
Specifies the other graph, of the same type as *self*, with which to intersect.

nodes_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate node attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps node attributes to functions. The default is None.

edges_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate edge attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps edge attributes to functions. The default is None.

Returns

SAS graph object
Returns a graph that is constructed by the intersection of the two graphs.

is_biconnected

```
is_biconnected()
```

Specifies whether or not the input graph is biconnected.

Returns

bool
Indicates whether the input graph is biconnected (True) or not (False).

is_bipartite

```
is_bipartite()
```

Determines whether the input graph is bipartite.

Returns

bool
Indicates whether the input graph is bipartite.

is_connected

```
is_connected()
```

Specifies whether or not the input graph is connected.

Returns

bool
Indicates whether the input graph is connected (True) or not (False).

is_cycle

```
is_cycle()
```

Determines whether the input graph is a simple cycle.

Returns

bool
Indicates whether the input graph is a simple cycle (True) or not (False).

is_dag

```
is_dag()
```

Determines whether the input graph is a directed acyclic graph (DAG).

Returns

bool
Indicates whether the input graph is a directed acyclic graph (DAG).

is_directed

```
is_directed()
```

Specifies whether or not the graph is directed.

Returns

bool
Indicates whether or not the graph is directed.

is_hamiltonian

```
is_hamiltonian(maxtime=None)
```

Determines whether the graph is Hamiltonian (has a Hamiltonian cycle) or not.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time to determine whether or not the graph is Hamiltonian. The default is None.

Returns

bool or None
Indicates whether or not the input graph is determined to be Hamiltonian. Returns None if no determination is made.

is_immutable

```
is_immutable()
```

Specifies whether or not the graph is immutable.

Returns

bool
Indicates whether or not the graph is immutable.

is_multigraph

```
is_multigraph()
```

Specifies whether or not the graph allows multiedges.

Returns

bool
Indicates whether or not the graph allows multiedges.

is_path

```
is_path()
```

Determines whether the input graph is a simple path.

Returns

bool
Indicates whether the input graph is a simple path (True) or not (False).

is_strongly_connected

```
is_strongly_connected()
```

Specifies whether the input graph is strongly connected.

Returns

bool
Indicates whether the input graph is strongly connected (True) or not (False).

is_weakly_connected

```
is_weakly_connected()
```

Specifies whether the input graph is weakly connected.

Returns

bool
Indicates whether the input graph is weakly connected (True) or not (False).

label_propagation

```
label_propagation(edge_weight=None, maxiters=100, tolerance=0.05, *,
data=True)
```

Computes the communities of the graph by using the label propagation algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of label changes for all nodes in the graph is less than the value. The default is 0.05.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one community. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

label_propagation_nodes

```
label_propagation_nodes(edge_weight=None, maxiters=100, tolerance=0.05, *,
data=True, inplace=False)
```

Computes community assignments for all nodes in the graph by using the label propagation algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of label changes for all nodes in the graph is less than the value. The default is 0.05.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

inplace : bool, optional
Specifies whether to add the community values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains the node attribute *community*, which denotes the community assignment for each node. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

leaf_node_count

```
leaf_node_count()
```

Computes a graph's leaf node count.

Returns

int
Returns the number of leaf nodes.

local_transitivity

```
local_transitivity(*, inplace=False)
```

Computes the local transitivity (clustering coefficient) and stores the results as node attributes on the graph.

Parameters

inplace : bool, optional
Specifies whether to add the transitivity values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains node attribute 'transitivity'. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

louvain

```
louvain(*, edge_weight=None, resolution=1, tolerance=0.001, maxiters=100,
data=True)
```

Computes the communities of the graph by using the Louvain algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

resolution : float, optional
Specifies the factor in the modularity equation that affects the expected number of links between two nodes. A higher value produces more communities. The default is 1.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of modularity gain between two consecutive iterations is less than the value. The default is 0.001.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one community. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

louvain_nodes

```
louvain_nodes(*, edge_weight=None, resolution=1, tolerance=0.001,
maxiters=100, data=True, inplace=False)
```

Computes community assignments for all nodes in the graph by using the Louvain algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

resolution : float, optional
Specifies the factor in the modularity equation that affects the expected number of links between two nodes. A higher value produces more communities. The default is 1.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of modularity gain between two consecutive iterations is less than the value. The default is 0.001.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

inplace : bool, optional
Specifies whether to add the community values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains the node attribute *community*, which denotes the community assignment for each node. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

maxflow

```
maxflow(source, sink, capacity_attr='upper', *, inplace=False, data=True)
```

Calculates the maximum flow from a source node to a sink node in a graph.

Parameters

source : number or str
Specifies the source node for the maximum flow.

sink : number or str
Specifies the sink node for the maximum flow.

capacity_attr : str, optional
Specifies the name of the edge attribute that contains the capacity of the edge. The default is 'upper'.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute *flow*, which contains the flow values. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, the returned graph contains only the flow edges.

maximal_cliques

```
maximal_cliques(maxcliques='ALL', node_weight=None, edge_weight=None,
minsize=1, maxsize=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, order_by=None, *,
data=True, ascending=False)
```

Enumerates all maximal cliques of the graph.

Parameters

maxcliques : int or 'ALL', optional
Specifies the maximum number of enumerated cliques to return. The default is 'ALL'.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

minsize : int, optional
Specifies the minimum number of nodes in a clique. The default is 1.

maxsize : int or None, optional
Specifies the maximum number of nodes in a clique. The default is None.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a clique. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a clique. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a clique. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a clique. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds for enumeration to spend. The default is None.

order_by : 'node' or None, optional
Specifies whether to sort the cliques by the number of nodes within each clique. The default is None, which means that the result is not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one maximal clique. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

min_cut

```
min_cut(edge_weight=None, *, inplace=False, data=True)
```

Calculates a minimum cut of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted minimum cut is calculated.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute 'mincut', which indicates the edges to include in the minimum cut set, and the node attribute 'mincut_partition', which indicates the node partition of the calculated minimum cut. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, a modified copy of the graph is returned.

min_s_t_cut

```
min_s_t_cut(source, sink, edge_weight=None, *, inplace=False, data=True)
```

Calculates a minimum s-t cut of a graph.

Parameters

source : number or str
Specifies the source node (s) for the minimum s-t cut.

sink : number or str
Specifies the sink node (t) for the minimum s-t cut.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted minimum s-t cut is calculated.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute *s_t_cut*, which indicates the edges to include in the minimum s-t cut set; and the node attribute *s_t_cut_partition*, which indicates the node partition of the calculated minimum s-t cut. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, a modified copy of the graph is returned.

mincostflow

```
mincostflow(demand_lower_attr='lower', demand_upper_attr='upper',
capacity_lower_attr='lower', capacity_upper_attr='upper', cost_attr='weight',
maxtime=None, *, inplace=False, data=True)
```

Calculates the minimum-cost network flow for the graph.

Parameters

demand_lower_attr : str, optional
Specifies the name of the node attribute that contains the lower bound of the node supply. The default is 'lower'.

demand_upper_attr : str, optional
Specifies the name of the node attribute that contains the upper bound of the node supply. The default is 'upper'.

capacity_lower_attr : str, optional
Specifies the name of the edge attribute that contains the capacity lower bound of the edge. The default is 'lower'.

capacity_upper_attr : str, optional
Specifies the name of the edge attribute that contains the capacity upper bound of the edge. The default is 'upper'.

cost_attr : str, optional
Specifies the name of the edge attribute that contains the edge cost. The default is 'weight'.

maxtime : float or None, optional
Specifies the maximum amount of time to allow for computing the minimum-cost network flow. The default is None.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attributes *flow* and *reduced_cost*, which contain the flow and reduced cost values, respectively; and the node attribute *dual*, which contains the optimal dual value. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned.

minimum_spanning_tree

```
minimum_spanning_tree(edge_weight=None, source=None, *, inplace=False,
data=True)
```

Calculates a minimum spanning tree (MST) or forest of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

source : number or str or None
Specifies the source node for a directed MST. The default is None.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute 'mst', which indicates the edges to include in the spanning tree or forest. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, the returned graph contains only the spanning tree or forest edges.

minimum_weight_full_matching

```
minimum_weight_full_matching(edge_weight=None, *, data=True)
```

Returns a minimum-weight full matching of the bipartite graph *self*.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns a directed graph whose edges identify the assignment mapping in the minimum-weight full matching solution. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

neighbors

```
neighbors(n)
```

Returns an iterator over all successor nodes of n.

Parameters

n : node
Specifies a node in the graph.

Returns

iterator
Returns an iterator over successors of node n.

new_edge_key

```
new_edge_key(u, v)
```

Returns an unused key for the edges between nodes *u* and *v*.

Parameters

u : numeric or str
Specifies the *from* node to add to the graph.

v : numeric or str
Specifies the *to* node to add to the graph.

Returns

int
Returns a unique edge key value.

node_clique_numbers

```
node_clique_numbers(*, inplace=False, data=True)
```

Computes node clique numbers and stores the results as attributes on the graph.

Parameters

inplace : bool, optional
Specifies whether to add the clique number values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the value of the *inplace* parameter is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph or modifies the existing graph to contain the node attribute *clique_number*, which includes the clique number of each node. The graph g has the attribute g.solution_summary, which contains information about the results of the algorithm.

number_of_edges

```
number_of_edges(u=None, v=None)
```

Returns the number of edges in the graph.

When you specify *u* and *v*, the number of edges between *u* and *v* is returned. When you specify *u* but *v* is None, the number of connections from *u* to other nodes (out-degree of node *u*) is returned. When you specify *v* but *u* is None, the number of connections from other nodes to *v* (in-degree of node *v*) is returned.

Parameters

u : numeric or str or None, optional
Specifies the *from* node. The default is None.

v : numeric or str or None, optional
Specifies the *to* node. The default is None.

Returns

int
Returns the number of edges in the graph. When you specify *u* and *v*, the number of edges between *u* and *v* is returned. When you specify *u* but *v* is None, the number of connections from *u* to other nodes (out-degree of node *u*) is returned. When you specify *v* but *u* is None, the number of connections from other nodes to *v* (in-degree of node *v*) is returned.

number_of_nodes

```
number_of_nodes()
```

Returns the number of nodes in the graph.

Returns

int
Returns the number of nodes in the graph.

out_degree

```
out_degree(n=None, edge_weight=None)
```

Returns the out-degree of nodes.

Parameters

n : single node or container of nodes or None, optional
Specifies which nodes to include. The default is None, which means that the out-degree of all nodes is returned.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None.

Returns

numeric or dictionary
Returns the out-degree of the node if a single node is passed, or returns a dictionary with nodes as keys and their out-degrees as values if None or a list of nodes is passed.

predecessors

```
predecessors(n)
```

Returns an iterator over all predecessor nodes of n.

Parameters

n : node
Specifies a node in the graph.

Returns

iterator
Returns an iterator over predecessors of node n.

projected_graph

```
projected_graph(nodes, neighbors=None, directed_method=None,
_measure=None, *, multigraph=False, data=True)
```

Computes a projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in self other than *nodes*.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used. When the *directed_method* parameter is specified for an undirected graph, a ValueError is raised.

multigraph : bool, optional
Specifies whether or not to return a multigraph. When the value is True, the algorithm returns a multigraph in which the multiple edges represent multiple shared neighbors. The edge attribute 'neighbor' stores the label of the shared neighbor. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

projected_graph_adamic_adar

```
projected_graph_adamic_adar(nodes, neighbors=None, directed_method=None,
*, data=True)
```

Computes an Adamic-Adar projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than *nodes*.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'adamic_adar' edge attribute.

projected_graph_common_neighbors

```
projected_graph_common_neighbors(nodes, neighbors=None,
directed_method=None, *, data=True)
```

Computes a common neighbors projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'common_neighbors' edge attribute.

projected_graph_cosine

```
projected_graph_cosine(nodes, edge_weight=None, neighbors=None,
directed_method=None, *, data=True)
```

Computes a cosine projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'cosine' edge attribute.

projected_graph_dice

```
projected_graph_dice(nodes, neighbors=None, directed_method=None, *,
data=True)
```

Computes a Sørensen-Dice projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the subset of nodes that are specified by the *neighbors* parameter. The strength of association in the calculated projection is represented by the 'dice' edge attribute.

projected_graph_jaccard

```
projected_graph_jaccard(nodes, neighbors=None, directed_method=None, *,
data=True)
```

Computes a Jaccard projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'jaccard' edge attribute.

query

```
query(q, expr=None, _node_attr_to_ignore=None, _edge_attr_to_ignore=None, *,
induced=False, break_symmetry=False, _generate_output=True,
_orbits_only=False, **kwargs)
```

Queries for the subgraph isomorphisms of q within g.

Parameters

q : graph
Specifies the query graph.

expr : _Expression or None, optional
Specifies additional logic to filter the query results. The default is None.

induced : bool, optional
Specifies whether or not to return only node-induced matches. The default is False.

break_symmetry : bool, optional
Specifies whether or not to break symmetry and eliminate automorphic relationships between matches. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one isomorphic mapping from *q* to *self*. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

reach_graph

```
reach_graph(source, max_reach=1, *, data=True)
```

Computes the reach graph for the source or set of sources.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to compute the reach network.

max_reach : int, optional
Specifies the number of steps (or hops) to traverse from the source nodes. The default is 1.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns a graph that represents the reach network. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

reach_graph_per_source

```
reach_graph_per_source(source, max_reach=1, *, data=True)
```

Computes a separate reach graph for each given source node.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to compute the reach network.

max_reach : int, optional
Specifies the number of steps (or hops) to traverse from the source nodes. The default is 1.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one reach network. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

remove_edge

```
remove_edge(u, v, key=None)
```

Removes a single edge from the graph.

Parameters

u : numeric or str
Specifies the *from* node.

v : numeric or str
Specifies the *to* node.

key : numeric or str or None
Specifies the edge key. The default is None, which means that when there are multiple edges between u and v, the most recently added edge is removed.

Examples

```
G.remove_edge(1,2)
```

remove_edges_from

```
remove_edges_from(edges)
```

Removes the edges from the graph.

Parameters

edges : an iterable container of edges
Specifies the edges to remove from the graph. When you specify two-tuples (u, v), the most recently added edge between u and v is removed. When you specify three-tuples (u, v, key) or four-tuples (u, v, key, data), the edge between u and v with the specified key is removed.

remove_isolated_nodes

```
remove_isolated_nodes(immutable=None, *, inplace=False)
```

Removes isolated nodes from a graph.

Parameters

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the input graph. The default is False.

Returns

SAS graph object or None
Returns the resulting graph after isolated nodes are removed, or returns nothing if the *inplace* parameter value is True.

remove_node

```
remove_node(node)
```

Removes a single node from the graph.

Parameters

node : numeric or str
Specifies the node to remove from the graph.

remove_nodes_from

```
remove_nodes_from(nodes)
```

Removes nodes from the graph. If a node does not exist in the graph, it is silently ignored.

Parameters

nodes : an iterable container of nodes
Specifies the nodes to remove from the graph.

reverse

```
reverse(*, data=True)
```

Returns a copy of a directed graph with reversed edges.

Parameters

data : bool, optional
Specifies whether or not to retain attributes. The default is True, meaning that the node and edge attributes of the graph are retained.

Returns

directed graph object
Returns a directed graph that holds the reversed edges.

set_edge_attributes

```
set_edge_attributes(values, name=None, immutable=None, *, inplace=False)
```

Sets edge attributes by using a given value or dictionary of values.

Parameters

values : scalar or dict-like or Pandas DataFrame
Specifies what the edge attribute should be set to. When this parameter is set to a Pandas DataFrame, it is expected to have from, to, and edge_key (only for multigraphs) columns, and the other columns are treated as edge attributes to add. When *values* is a dictionary, the keys are edges and the values are either edge attribute values or dictionaries of attribute name-value pairs. For multigraphs, the edge tuples must be of the form (u, v, key). For simple graphs, the keys must be tuples of the form (u, v). When this parameter is not set to a dictionary, it is treated as a single attribute value that is then applied to every edge in *self*. The attribute name is specified by the *name* parameter.

name : str or None, optional
Specifies the name of the edge attribute to set if the *values* parameter is set to a scalar. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is False, a copy of the input graph with the new edge attributes is returned. If the *inplace* parameter value is True, the graph is modified by setting the specified edge attributes.

set_node_attributes

```
set_node_attributes(values, name=None, immutable=None, *, inplace=False)
```

Sets node attributes from a given value or dictionary of values.

Parameters

values : scalar or dict-like or Pandas DataFrame
Specifies what the node attribute should be set to. When *values* is not a dictionary, it is treated as a single attribute value that is then applied to every node in *self*. This means that if you provide a mutable object, such as a list, updates to that object are reflected in the node attribute for every node. The attribute name is specified by the *name* parameter. When *values* is a dictionary, the keys are nodes and the values are either node attribute values or dictionaries of attribute name-value pairs.

name : str or None, optional
Specifies the name of the node attribute to set if the *values* parameter is set to a scalar. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not the graph is modified in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by setting the specified node attributes. If the *inplace* parameter value is False, a copy of the input graph with the new node attributes is returned.

shortest_path_distances

```
shortest_path_distances(source=None, sink=None, edge_weight=None,
minedgewt=None, maxedgewt=None)
```

Calculates the shortest path distances in a graph.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) and restricts enumeration to the shortest paths that contain the specified source node(s). The default is None.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) and restricts enumeration to the shortest paths that contain the specified sink node(s). The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

Returns

dict
Returns a dictionary where each key is a (source, sink) tuple and its value is the corresponding shortest source-sink path distance.

similarity_adamic_adar

```
similarity_adamic_adar(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Adamic-Adar node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_common_neighbors

```
similarity_common_neighbors(source, sink=None, top_k=None, bottom_k=None,
*, exclude_source=False, exclude_existing_neighbors=False)
```

Computes common neighbors node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_cosine

```
similarity_cosine(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes cosine node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_dice

```
similarity_dice(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Sørensen-Dice node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_graph

```
similarity_graph(measure, min_score=2.220446049250313e-16, max_score=inf,
*, exclude_source=False, exclude_existing_neighbors=False, data=True,
**kwargs)
```

Computes a node similarity graph.

Parameters

measure : str, one of {'adamic_adar', 'common_neighbors', 'cosine', 'dice', 'jaccard'}
Specifies the metric to be used to compute the node similarity.

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the edge attribute that corresponds to the measure name. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_adamic_adar

```
similarity_graph_adamic_adar(min_score=2.220446049250313e-16,
max_score=inf, *, exclude_source=False, exclude_existing_neighbors=False,
data=True)
```

Computes an Adamic-Adar node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'adamic_adar' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_common_neighbors

```
similarity_graph_common_neighbors(min_score=2.220446049250313e-16,
max_score=inf, *, exclude_source=False, exclude_existing_neighbors=False,
data=True)
```

Computes a common neighbors node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'common_neighbors' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_cosine

```
similarity_graph_cosine(edge_weight=None,
min_score=2.220446049250313e-16, max_score=inf, *, exclude_source=False,
exclude_existing_neighbors=False, data=True)
```

Computes a cosine node similarity graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'cosine' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_dice

```
similarity_graph_dice(min_score=2.220446049250313e-16, max_score=inf, *,
exclude_source=False, exclude_existing_neighbors=False, data=True)
```

Computes a Sørensen-Dice node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the *dice* edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_jaccard

```
similarity_graph_jaccard(min_score=2.220446049250313e-16, max_score=inf, *,
exclude_source=False, exclude_existing_neighbors=False, data=True)
```

Computes a Jaccard node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'jaccard' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_jaccard

```
similarity_jaccard(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Jaccard node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

singleton_node_count

```
singleton_node_count()
```

Computes a graph's singleton node count.

Returns

int
Returns the number of singleton nodes.

successors

```
successors(n)
```

Returns an iterator over all successor nodes of n.

Parameters

n : node
Specifies a node in the graph.

Returns

iterator
Returns an iterator over successors of node n.

to_directed

```
to_directed()
```

Returns a directed copy of the input graph.

Returns

SAS graph object
Returns a directed copy of the input graph.

to_immutable

```
to_immutable()
```

Creates an immutable graph from the input graph.

Returns

SAS graph object
Returns an immutable SAS graph object.

to_mutable

```
to_mutable()
```

Creates a mutable graph from the input graph.

Returns

SAS graph object
Returns a mutable SAS graph object.

to_pandas_edges

```
to_pandas_edges(*, data=True, _for_conversion_to_immutable=False)
```

Returns the graph edges as a Pandas DataFrame.

Parameters

data : bool or list, optional
Specifies whether or not to retain attributes. The default is True. When the value is True (or when a list is given), the Pandas DataFrame includes all edge attributes (or all listed edge attributes).

Returns

dataframe
Returns a Pandas DataFrame that includes the graph edges.

to_pandas_nodes

```
to_pandas_nodes(*, data=True, _for_conversion_to_immutable=False)
```

Returns the graph nodes as a Pandas DataFrame.

Parameters

data : bool or list, optional
Specifies whether or not to retain attributes. The default is True. When the value is True (or when a list is given), the Pandas DataFrame includes all the node attributes (or all the listed attributes).

Returns

dataframe
Returns a Pandas DataFrame that includes the graph nodes.

to_undirected

```
to_undirected()
```

Returns a directed copy of the input graph.

Returns

SAS graph object
Returns a directed copy of the input graph.

topological_ordering

```
topological_ordering()
```

Finds a topological ordering of the graph's nodes.

Yields

generator
Yields a generator of nodes in topological order. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

transitive_closure

```
transitive_closure()
```

Calculates the transitive closure of a graph.

Returns

SAS graph object
Returns the transitive closure of the input graph that is returned as a SAS MultiGraph or MultiDiGraph object. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

triangle_count

```
triangle_count()
```

Computes a graph's triangle count.

Returns

int
Returns the number of triangles.

tsp_tour

```
tsp_tour(edge_weight=None, maxtime=None, minobjective=None, *,
inplace=False, use_milp=True, data=True, **options)
```

Calculates a traveling salesman problem (TSP) tour of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted TSP is calculated.

maxtime : float or None, optional
Specifies the maximum amount of time for solving a TSP. The default is None.

minobjective : float, optional
Specifies a minimum value such that when a solution is found whose objective is less than or equal to this value, the algorithm stops.

inplace : bool, optional
Specifies whether to add the TSP order values as attributes to the input graph directly or to create a copy of the graph. The default is False.

use_milp : bool, optional
Specifies whether or not to use a mixed integer linear programming (MILP) solver to solve a TSP. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is False, the returned graph is a Cycle that contains the calculated TSP tour edges. When the *inplace* parameter value is True, the graph is modified in place and None is returned. The modified graph contains the edge attributes *tsp* and *tsp_order* and the node attribute *tsp_order*.

union

```
union(rhs, nodes_attrs_agg=None, edges_attrs_agg=None)
```

Returns the union of input graphs by finding the unions of their nodes and edges.

The node attributes and edge attributes are aggregated using the rules that are defined by the *nodes_attrs_agg* and *edges_attrs_agg* parameters.

Parameters

rhs : SAS graph object
Specifies the other graph, of the same type as *self*, with which to find the union.

nodes_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate node attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps node attributes to functions. The default is None.

edges_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate edge attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps edge attributes to functions. The default is None.

Returns

SAS graph object
Returns a graph that is constructed by the union of the two graphs.

vrp

```
vrp(capacity, depot, node_demand='demand', edge_cost='weight', minroutes=1,
maxroutes=None, maxtime=None, minobjective=None, *, use_milp=True,
data=True, **options)
```

Solves the vehicle routing problem.

Parameters

capacity : float
Specifies the capacity of each vehicle.

depot : node
Specifies the depot node for the vehicle routing problem.

node_demand : str or dict or iterable
Specifies the quantity of goods to deliver to or pick up from a customer. The default is 'demand'.

edge_cost : str or dict or iterable
Specifies the cost of traversing each edge. The default is 'weight'.

minroutes : int, optional
Specifies the minimum number of routes that are allowed to service demand.

maxroutes : int, optional
Specifies the maximum number of routes that are allowed to service demand.

maxtime : float or None, optional
Specifies the maximum amount of time for solving the vehicle routing problem. The default is None.

minobjective : float, optional
Specifies a minimum value such that when a solution is found whose objective is less than or equal to this value, the algorithm stops.

use_milp : bool, optional
Specifies whether or not to use a mixed integer linear programming (MILP) solver to solve the vehicle routing problem. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Cycle objects; each cycle represents one route. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

vrptw

```
vrptw(capacity, depot, node_demand='demand', edge_cost='weight',
travel_time='weight', timewindow_lower=None, timewindow_upper=None,
service_time=None, minroutes=1, maxroutes=None, maxtime=None, *,
data=True)
```

Solves the vehicle routing problem with time windows.

Parameters

capacity : float
Specifies the capacity of each vehicle.

depot : node
Specifies the depot node for the vehicle routing problem.

node_demand : str or dict or iterable
Specifies the quantity of goods to deliver to or pick up from a customer. The default is 'demand'.

edge_cost : str or dict or iterable
Specifies the cost of traversing each edge.

travel_time : str or dict or iterable
Specifies attributes or values that define the time that it takes to traverse each edge. The default is 'weight'.

timewindow_lower : str or dict or iterable or None, optional
Specifies attributes or values that define the lower time limits within which a delivery vehicle can arrive at the customer nodes. The default is None.

timewindow_upper : str or dict or iterable or None, optional
Specifies attributes or values that define the upper time limits within which a delivery vehicle can arrive at the customer nodes. The default is None.

service_time : str or dict or iterable or None, optional
Specifies attributes or values that define the service time of the nodes. The default is None.

minroutes : int, optional
Specifies the minimum number of routes that are allowed to service demand.

maxroutes : int, optional
Specifies the maximum number of routes that are allowed to service demand.

maxtime : float or None, optional
Specifies the maximum amount of time for solving the vehicle routing problem. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Cycle objects; each cycle represents one route. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

MultiGraph

Base class for undirected multigraphs.

API: MultiGraph

class sasviya.network.classes.multigraph.MultiGraph(*args, **kwargs)

Self-loops and multiedges are allowed.

Attributes

edges
Returns the edges of the graph.

nodes
Returns the nodes of the graph.

Methods

Method Name	Description
add_core_number_attr	Computes the core numbers and stores the results as node attributes on the graph.
add_edge	Adds a single edge to the graph.
add_edges_from	Adds edges to the graph.
add_node	Adds a node to the graph.
add_nodes_from	Adds the specified nodes to the graph.
add_topological_order_attr	Computes a topological ordering and stores the results as node attributes of the graph.
add_weighted_edges_from	Adds weighted edges to the graph.

Method Name	Description
adjacency	Returns an iterator over (node, adjacency dict) for all nodes.
allows_multiedges	Specifies whether or not the graph allows multiedges.
allows_selfloops	Specifies whether or not the graph allows self-loops.
apply	Applies a function to all node or edge attributes and updates them.
approximate_diameter	Computes an approximation of the graph's diameter.
articulation_point_count	Computes a graph's articulation point count.
articulation_points	Finds the articulation points (cut points) of a graph.
assortativity_degree	Computes a graph's degree assortativity.
assortativity_nominal	Computes a graph's nominal assortativity.
assortativity_numeric	Computes a graph's numeric assortativity.
average_shortest_path	Computes a graph's average shortest path.
biconcomp_distribution_edges	Computes the distribution of the number of edges in each biconnected components by using *numpy.histogram*.
biconcomp_distribution_nodes	Computes the distribution of the number of nodes in each biconnected components by using *numpy.histogram*.
biconcomp_size_edges	Computes the number of edges within each biconnected component.
biconcomp_size_nodes	Computes the number of nodes within each biconnected component.
biconnected_component_count	Computes a graph's biconnected component count.
biconnected_components	Computes the graph's biconnected components for an undirected graph.
block_cut_tree	Computes the block-cut tree of a graph.
centrality_betweenness	Computes weighted or unweighted betweenness centrality and stores the results as attributes on the graph.

Method Name	Description
centrality_closeness	Computes weighted or unweighted closeness centrality and stores the results as attributes on the graph.
centrality_degree	Computes weighted or unweighted degree centrality and stores the results as attributes on the graph.
centrality_eigenvector	Computes weighted or unweighted eigenvector centrality and stores the results as attributes on the graph.
centrality_hub_authority	Computes weighted or unweighted hub and authority centrality and stores the results as attributes on the graph.
centrality_influence	Computes weighted or unweighted influence centrality and stores the results as attributes on the graph.
centrality_pagerank	Computes weighted or unweighted PageRank centrality and stores the results as attributes on the graph.
clear	Removes all nodes and edges from the graph.
clear_edges	Removes all edges from the graph but retains nodes.
concomp_distribution_edges	Computes the distribution of the number of edges in each connected component by using *numpy.histogram*.
concomp_distribution_nodes	Computes the distribution of the number of nodes in each connected component by using *numpy.histogram*.
concomp_size_edges	Computes the number of edges within each connected component.
concomp_size_nodes	Computes the number of nodes within each connected component.
connected_component_count	Computes a graph's connected component count.
connected_components	Computes the connected components of a graph.
copy	Returns a copy of the input graph.
core_decomposition	Computes a core decomposition of the graph.

Method Name	Description
cycle_count	Counts the number of cycles in a graph.
degree	Returns the degree of nodes.
density	Computes the density of the graph.
diameter	Computes a graph's diameter.
drop_edge_attributes	Removes edge attributes if they exist.
drop_node_attributes	Removes node attributes if they exist.
enumerate_cycle_edges	Enumerates the cycles in a graph and returns the edges of the cycles.
enumerate_cycle_nodes	Enumerates the cycles in a graph and returns the nodes of the cycles.
enumerate_cycles	Enumerates the cycles in a graph and returns the cycle graph objects.
enumerate_path_edges	Enumerates the paths in a graph and returns the edges of the paths.
enumerate_path_nodes	Enumerates the paths in a graph and returns the nodes of the paths.
enumerate_paths	Enumerates the paths in a graph and returns the Path graph objects.
enumerate_sequence_shortest_paths	Enumerates the shortest paths in a graph that visit the indicated sequence nodes in order.
enumerate_shortest_path_edges	Enumerates the shortest paths in a graph and returns the edges of the paths.
enumerate_shortest_path_nodes	Enumerates the shortest paths in a graph and returns the nodes of the paths.
enumerate_shortest_paths	Enumerates the shortest paths in a graph and returns the Path graph objects.
get_meta_graph	Constructs a metagraph from the input graph.
get_nlargest_biconcomps	Returns a list of up to *n* largest biconnected components.
get_nlargest_concomps	Returns a list of up to *n* largest connected (with highest number of nodes) components of a graph.

Method Name	Description
has_edge_attribute	Checks the edges of the input graph for the specified edge attribute.
has_node_attribute	Checks the nodes of the input graph for the specified node attribute.
intersection	Returns the intersection of input graphs by finding the intersections of their nodes and edges.
is_biconnected	Specifies whether or not the input graph is biconnected.
is_bipartite	Determines whether the input graph is bipartite.
is_connected	Specifies whether or not the input graph is connected.
is_cycle	Determines whether the input graph is a simple cycle.
is_dag	Determines whether the input graph is a directed acyclic graph (DAG).
is_directed	Specifies whether or not the graph is directed.
is_hamiltonian	Determines whether the graph is Hamiltonian (has a Hamiltonian cycle) or not.
is_immutable	Specifies whether or not the graph is immutable.
is_multigraph	Specifies whether or not the graph allows multiedges.
is_path	Determines whether the input graph is a simple path.
is_strongly_connected	Specifies whether the input graph is strongly connected.
is_weakly_connected	Specifies whether the input graph is weakly connected.
label_propagation	Computes the communities of the graph by using the label propagation algorithm.
label_propagation_nodes	Computes community assignments for all nodes in the graph by using the label propagation algorithm.
leaf_node_count	Computes a graph's leaf node count.

Method Name	Description
local_transitivity	Computes the local transitivity (clustering coefficient) and stores the results as node attributes on the graph.
louvain	Computes the communities of the graph by using the Louvain algorithm.
louvain_nodes	Computes community assignments for all nodes in the graph by using the Louvain algorithm.
maxflow	Calculates the maximum flow from a source node to a sink node in a graph.
maximal_cliques	Enumerates all maximal cliques of the graph.
min_cut	Calculates a minimum cut of a graph.
min_s_t_cut	Calculates a minimum s-t cut of a graph.
mincostflow	Calculates the minimum-cost network flow for the graph.
minimum_spanning_tree	Calculates a minimum spanning tree (MST) or forest of a graph.
minimum_weight_full_matching	Returns a minimum-weight full matching of the bipartite graph *self*.
neighbors	Returns an iterator over all neighbors of node n.
new_edge_key	Returns an unused key for the edges between nodes *u* and *v*.
node_clique_numbers	Computes node clique numbers and stores the results as attributes on the graph.
number_of_edges	Returns the number of edges in the graph.
number_of_nodes	Returns the number of nodes in the graph.
projected_graph	Computes a projected graph.
projected_graph_adamic_adar	Computes an Adamic-Adar projected graph.
projected_graph_common_neighbors	Computes a common neighbors projected graph.
projected_graph_cosine	Computes a cosine projected graph.
projected_graph_dice	Computes a Sørensen-Dice projected graph.

Method Name	Description
projected_graph_jaccard	Computes a Jaccard projected graph.
query	Queries for the subgraph isomorphisms of q within g.
reach_graph	Computes the reach graph for the source or set of sources.
reach_graph_per_source	Computes a separate reach graph for each given source node.
remove_edge	Removes a single edge from the graph.
remove_edges_from	Removes edges from the graph.
remove_isolated_nodes	Removes isolated nodes from a graph.
remove_node	Removes a single node from the graph.
remove_nodes_from	Removes nodes from the graph. If a node does not exist in the graph, it is silently ignored.
set_edge_attributes	Sets edge attributes by using a given value or dictionary of values.
set_node_attributes	Sets node attributes from a given value or dictionary of values.
shortest_path_distances	Calculates the shortest path distances in a graph.
similarity_adamic_adar	Computes Adamic-Adar node similarity.
similarity_common_neighbors	Computes common neighbors node similarity.
similarity_cosine	Computes cosine node similarity.
similarity_dice	Computes Sørensen-Dice node similarity.
similarity_graph	Computes a node similarity graph.
similarity_graph_adamic_adar	Computes an Adamic-Adar node similarity graph.
similarity_graph_common_neighbors	Computes a common neighbors node similarity graph.
similarity_graph_cosine	Computes a cosine node similarity graph.
similarity_graph_dice	Computes a Sørensen-Dice node similarity graph.
similarity_graph_jaccard	Computes a Jaccard node similarity graph.

Method Name	Description
similarity_jaccard	Computes Jaccard node similarity.
singleton_node_count	Computes a graph's singleton node count.
to_directed	Returns a directed copy of the input graph.
to_immutable	Creates an immutable graph from the input graph.
to_mutable	Creates a mutable graph from the input graph.
to_pandas_edges	Returns the graph edges as a Pandas DataFrame.
to_pandas_nodes	Returns the graph nodes as a Pandas DataFrame.
to_undirected	Returns a directed copy of the input graph.
topological_ordering	Finds a topological ordering of the graph's nodes.
transitive_closure	Calculates the transitive closure of a graph.
triangle_count	Computes a graph's triangle count.
tsp_tour	Calculates a traveling salesman problem (TSP) tour of a graph.
union	Returns the union of input graphs by finding the unions of their nodes and edges.
vrp	Solves the vehicle routing problem.
vrptw	Solves the vehicle routing problem with time windows.

add_core_number_attr

```
add_core_number_attr(maxtime=None, *, inplace=False, data=True)
```

Computes the core numbers and stores the results as node attributes on the graph.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time for core decomposition to spend. The default is None.

inplace : bool, optional
Specifies whether to add the core numbers as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph with core numbers that are stored as a node attribute named *core*. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

add_edge

```
add_edge(u, v, key=None, **kwargs)
```

Adds a single edge to the graph.

Parameters

u : numeric or str
Specifies the *from* node.

v : numeric or str
Specifies the *to* node.

key : numeric or str
Specifies the key of the edge to add to the graph. The default is None.

Returns

numeric or str
Returns the key of the edge that is added to the graph.

Examples

```
G.add_edge(1,2,key='A',weight=3,type='parent')
```

add_edges_from

```
add_edges_from(edges, **kwargs)
```

Adds edges to the graph.

Parameters

edges : an iterable container of edges
Specifies the edges to add to the graph.

Returns

list
Returns the keys of the edges that are added to the graph.

Examples

```
G.add_edges_from([("A","B",0,{"weight":1,"color":"blue"}),
                  ("C","D",0,{"weight":2,"color":"red"})])
G.add_edges_from([["X","Y",0],["Y","Z",0], weight=1,color= "blue")
```

add_node

```
add_node(node, **attrs)
```

Adds a node to the graph.

Parameters

node : str or number or tuple
Specifies the ID of the node to add to the graph. If the value is a tuple, the first element should contain the node ID, and the second element is a dictionary that represents the node's attributes.

add_nodes_from

```
add_nodes_from(nodes, **attrs)
```

Adds the specified nodes to the graph.

Parameters

nodes : an iterable container of nodes
Specifies the nodes to add to the graph.

Examples

```
G.add_nodes_from([('X', {"weight":11,"color": "blue"}), ("Y", {"color": "blue"})])
G.add_nodes_from(('X', 'Y'), weight=1,color= "blue")
G.add_nodes_from(['X', 'Y'])
```

add_topological_order_attr

```
add_topological_order_attr(*, inplace=False, data=True)
```

Computes a topological ordering and stores the results as node attributes of the graph.

Parameters

inplace : bool, optional
Specifies whether or not to add the topological order values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a directed graph whose topological order is stored as a node attribute named *top_order*. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

add_weighted_edges_from

```
add_weighted_edges_from(edges, weight='weight', **kwargs)
```

Adds weighted edges to the graph.

Parameters

edges : an iterable container of edges
Specifies the weighted edges to add to the graph.

weight : str
Specifies the attribute name for the weighted edges to add. The default is 'weight'.

adjacency

```
adjacency()
```

Returns an iterator over (node, adjacency dict) for all nodes.

For directed graphs, only outgoing adjacencies are returned.

Returns

iterator
Returns an iterator over (node, adjacency dictionary) for all nodes in the graph.

allows_multiedges

```
allows_multiedges()
```

Specifies whether or not the graph allows multiedges.

Returns

bool
Indicates whether or not the graph allows multiedges.

allows_selfloops

```
allows_selfloops()
```

Specifies whether or not the graph allows self-loops.

Returns

bool
Indicates whether or not the graph allows self-loops.

apply

```
apply(node_attr_handler=None, edge_attr_handler=None, immutable=None, *,
inplace=False)
```

Applies a function to all node or edge attributes and updates them.

#use apply to fillna >>> sasnet.apply(G, edge_attr_handler=lambda data:{'w':0 if 'w' not in data else data['w']},inplace=True) >>> assert G.edges[(1,2)]['w'] == 0 >>> print(G.nodes(data=True)) #use apply to negate an attribute >>> sasnet.apply(G, node_attr_handler=lambda data:{'x': -data['x']},inplace=True) >>> assert G.nodes[1] ['x']== -10

Parameters

node_attr_handler : function or None, optional
Specifies the function to apply to node attributes. The default is None.

edge_attr_handler : function or None, optional
Specifies the function to apply to edge attributes. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the input graph. The default is False.

Returns

SAS graph object or None.
Returns the resulting graph after the function is applied, or returns nothing if the *inplace* parameter value is True.

Examples

```
import sasviya.network as sasnet
G = sasnet.Graph([[1,2,{}],[1,3,{'w':1}],[1,4,{'w':2}]])
sasnet.set_node_attributes(G,{1:10,2:20,3:30,4:40},name='x',inplace=True)
```

#use apply to fillna >>> sasnet.apply(G, edge_attr_handler=lambda data:{'w':0 if 'w' not in data else data['w']},inplace=True) >>> assert G.edges[(1,2)]['w'] == 0 >>> print(G.nodes(data=True)) #use apply to negate an attribute >>> sasnet.apply(G, node_attr_handler=lambda data:{'x': -data['x']},inplace=True) >>> assert G.nodes[1] ['x']== -10

approximate_diameter

```
approximate_diameter(edge_weight=None)
```

Computes an approximation of the graph's diameter.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted approximate diameter is computed.

Returns

float
Returns the approximate diameter.

articulation_point_count

```
articulation_point_count()
```

Computes a graph's articulation point count.

Returns

int
Returns the number of articulation points.

articulation_points

```
articulation_points()
```

Finds the articulation points (cut points) of a graph.

Returns

list
Returns a list of articulation points in the graph.

assortativity_degree

```
assortativity_degree(source_direction='out', target_direction='in',
edge_weight=None)
```

Computes a graph's degree assortativity.

Parameters

source_direction : {'in', 'out'}
Specifies the degree type (in-degree or out-degree) for the source node in a directed graph.

target_direction : {'in', 'out'}
Specifies the degree type (in-degree or out-degree) for the target node in a directed graph.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted assortativity degree is computed.

Returns

float
Returns the degree assortativity.

assortativity_nominal

```
assortativity_nominal(node_attr)
```

Computes a graph's nominal assortativity.

Parameters

node_attr : str
Specifies the name of the nominal node attribute to use.

Returns

float
Returns the nominal assortativity.

assortativity_numeric

```
assortativity_numeric(node_attr)
```

Computes a graph's numeric assortativity.

Parameters

node_attr : str
Specifies the name of the numeric node attribute to use.

Returns

float
Returns the numeric assortativity.

average_shortest_path

```
average_shortest_path(edge_weight=None)
```

Computes a graph's average shortest path.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted average shortest path is computed.

Returns

float
Returns the average shortest path.

biconcomp_distribution_edges

```
biconcomp_distribution_edges(bins=None)
```

Computes the distribution of the number of edges in each biconnected components by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is int (an integer), it specifies the number of equal-width bins. When *bins* is a list, it specifies bin ranges and should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is str (a string), it specifies the method to use for calculating the bin ranges. For example, 'stone','auto','doane','fd','rice','scott','sqrt','sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of biconnected components whose edge count is within the range of the bin.

biconcomp_distribution_nodes

```
biconcomp_distribution_nodes(bins=None)
```

Computes the distribution of the number of nodes in each biconnected components by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is int (an integer), it specifies the number of equal-width bins. When *bins* is a list, it specifies bin ranges and should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is str (a string), it specifies the method to use for calculating the bin ranges. For example, 'stone','auto','doane','fd','rice','scott','sqrt','sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of biconnected components whose node count is within the range of the bin.

biconcomp_size_edges

```
biconcomp_size_edges()
```

Computes the number of edges within each biconnected component.

Returns

dict
Returns a dictionary in which the keys are biconnected component IDs and the values are the number of edges within each component.

biconcomp_size_nodes

```
biconcomp_size_nodes()
```

Computes the number of nodes within each biconnected component.

Returns

dict
Returns a dictionary in which the keys are biconnected component IDs and the values are the number of nodes within each component.

biconnected_component_count

```
biconnected_component_count()
```

Computes a graph's biconnected component count.

Returns

int
Returns the number of biconnected components.

biconnected_components

```
biconnected_components(order_by=None, *, data=True, ascending=False)
```

Computes the graph's biconnected components for an undirected graph.

Parameters

order_by : 'node' or 'edge' or None, optional
Specifies the parameter that is used to sort the components by the number of nodes or edges within each component. The default is None, which means that the components are not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Returns the result as a generator of biconnected component subgraphs. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

block_cut_tree

```
block_cut_tree(*, return_nodes_map=False)
```

Computes the block-cut tree of a graph.

Parameters

return_nodes_map : bool, optional
Specifies whether or not to also compute the block-cut tree node map. The default is False.

Returns

SAS graph object
Returns a graph that represents the block-cut tree. When *return_nodes_map = True*, the output graph has the *nodes_map* attribute, which contains a dictionary that maps its nodes to a list of nodes in the original graph. Note: If a block is empty (that is, the corresponding biconnected component contains only articulation points), that block does not appear as a key in the *nodes_map* dictionary.

centrality_betweenness

```
centrality_betweenness(edge_weight=None, sample_percent=100, *,
inplace=False, data=True)
```

Computes weighted or unweighted betweenness centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that unweighted betweenness centrality is computed.

sample_percent : int, optional
Specifies the percentage of source nodes to sample for the approximate betweenness calculation. This parameter value should be in the range (0, 100]. The default is 100.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the node and edge attributes *between_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_closeness

```
centrality_closeness(edge_weight=None, nopath='diameter', *, inplace=False,
data=True)
```

Computes weighted or unweighted closeness centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted closeness centrality is computed.

nopath : {'diameter', 'harmonic', or 'zero'}, optional
Specifies a method to use for accounting for the shortest path distance between two nodes when a path does not exist. The valid values are as follows: - 'diameter' uses the graph diameter (plus one) as the shortest path distance between disconnected nodes. - 'harmonic' uses the harmonic formula for calculating closeness centrality. - 'zero' uses zero as the shortest path distance between disconnected nodes.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the closeness centrality node attributes *centr_close_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_degree

```
centrality_degree(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted degree centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted degree centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the degree centrality node attributes *centr_degree_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_eigenvector

```
centrality_eigenvector(edge_weight=None, maxiters=10000, *, inplace=False,
data=True)
```

Computes weighted or unweighted eigenvector centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted eigenvector centrality is computed.

maxiters : int, optional
Specifies the maximum number of iterations in order to limit the amount of computation time that is spent when convergence is slow. The default is 10,000.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the degree centrality node attributes *centr_eigen_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_hub_authority

```
centrality_hub_authority(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted hub and authority centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted hub and authority centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the hub and authority centrality node attributes *centr_hub_** and *centr_auth_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_influence

```
centrality_influence(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted influence centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted influence centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the first- and second-order influence centrality node attributes *centr_influence1_** and *centr_influence2_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_pagerank

```
centrality_pagerank(edge_weight=None, alpha=0.85, tolerance=1e-09, *,
inplace=False, data=True)
```

Computes weighted or unweighted PageRank centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted PageRank centrality is computed.

alpha : float, optional
Specifies the damping parameter for the PageRank centrality (or algorithm). The default is 0.85.

tolerance : float, optional
Specifies the tolerance parameter for the PageRank centrality (or algorithm). The default is 1E-9.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the PageRank centrality node attributes *centr_pagerank_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

clear

```
clear()
```

Removes all nodes and edges from the graph.

clear_edges

```
clear_edges()
```

Removes all edges from the graph but retains nodes.

concomp_distribution_edges

```
concomp_distribution_edges(bins=None, *, strongly=True)
```

Computes the distribution of the number of edges in each connected component by using *numpy.histogram*.

Parameters

bins : int or list or str, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is an integer (int), it represents the number of equal-width bins. When *bins* is a list, it specifies bin ranges and it should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)*

(inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is a string (str), it specifies the method to be used for calculating the bin ranges. For example, 'stone', 'auto', 'doane', 'fd', 'rice', 'scott', 'sqrt', 'sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of components whose edge count within the range of the bin.

concomp_distribution_nodes

```
concomp_distribution_nodes(bins=None, *, strongly=True)
```

Computes the distribution of the number of nodes in each connected component by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is an integer (int), it represents the number of equal-width bins. When *bins* is a list, it specifies bin ranges and it should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is a string (str), it specifies the method to use for calculating the bin ranges. For example, 'stone', 'auto', 'doane', 'fd', 'rice', 'scott', 'sqrt', 'sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of components whose node count is within the range of the bin.

concomp_size_edges

```
concomp_size_edges(*, strongly=True)
```

Computes the number of edges within each connected component.

Parameters

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary in which the keys are connected component IDs and the values are the number of edges within each component.

concomp_size_nodes

```
concomp_size_nodes(*, strongly=True)
```

Computes the number of nodes within each connected component.

Parameters

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary in which the keys are connected component IDs and the values are the number of nodes within each component.

connected_component_count

```
connected_component_count()
```

Computes a graph's connected component count.

Returns

int
Returns the number of connected components.

connected_components

```
connected_components(order_by=None, *, strongly=True, data=True,
ascending=False)
```

Computes the connected components of a graph.

Parameters

order_by : 'node' or 'edge' or None, optional
Specifies whether to sort the components by the number of nodes or edges within each component. The default is None, which means that the components are not sorted.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one connected component. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

copy

```
copy(immutable=None, *, data=True)
```

Returns a copy of the input graph.

Parameters

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, meaning that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns an independent copy of the input.

core_decomposition

```
core_decomposition(maxtime=None, *, data=True)
```

Computes a core decomposition of the graph.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time for core decomposition to spend. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one k-core in increasing core number. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

cycle_count

```
cycle_count(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Counts the number of cycles in a graph.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Returns

int
Returns a count of cycles in the graph.

degree

```
degree(n=None, edge_weight=None)
```

Returns the degree of nodes.

Parameters

n : single node or container of nodes or None, optional
Specifies which nodes to include. The default is None, which means that the degree of every node is returned.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None.

Returns

int or dictionary
Returns the degree of specified node(s).

density

```
density()
```

Computes the density of the graph.

Returns

float
Returns the density of the graph.

diameter

```
diameter(edge_weight=None)
```

Computes a graph's diameter.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted diameter is computed.

Returns

int or float
Returns the diameter.

drop_edge_attributes

```
drop_edge_attributes(attributes_subset=None, edges_subset=None,
immutable=None, *, inplace=False)
```

Removes edge attributes if they exist.

Parameters

attributes_subset : str or list, optional
Specifies the edge attribute or a list of the edge attributes to remove; for example, 'weight' or ['weight','color']. By default, all edge attributes are removed.

edges_subset : list or iterable of edges, optional
Specifies the edges to remove attributes from; for example, [(1,2),(2,3)]. By default the attributes for all edges are removed.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by removing the specified edge attributes. If the *inplace* parameter value is False, a copy of the input graph with the modified edge attributes is returned.

drop_node_attributes

```
drop_node_attributes(attributes_subset=None, nodes_subset=None,
immutable=None, *, inplace=False)
```

Removes node attributes if they exist.

Parameters

attributes_subset : str or list, optional
Specifies the node attribute or list of node attributes to remove; for example, ['weight','color']. By default, all node attributes are removed.

nodes_subset : list or iterable of nodes, optional
Specifies the nodes to drop attributes from; for example, [1,2,3] or (1,2,3) or [1] or (1,). By default, the attributes for all nodes are removed.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by removing the specified node attributes. If the *inplace* parameter value is False, a copy of the input graph with the modified node attributes is returned.

enumerate_cycle_edges

```
enumerate_cycle_edges(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Enumerates the cycles in a graph and returns the edges of the cycles.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Yields

generator
Yields a generator of cycles, each of which is given as a list of the cycles' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_cycle_nodes

```
enumerate_cycle_nodes(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Enumerates the cycles in a graph and returns the nodes of the cycles.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Yields

generator
Yields a generator of cycles, each of which is given as a list of the cycles' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_cycles

```
enumerate_cycles(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None, *,
data=True)
```

Enumerates the cycles in a graph and returns the cycle graph objects.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one cycle of the same type as the input graph. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_path_edges

```
enumerate_path_edges(source=None, sink=None, minlength=1,
maxlength=None, edge_weight=None, node_weight=None, minedgewt=None,
maxedgewt=None, minnodewt=None, maxnodewt=None, maxtime=None)
```

Enumerates the paths in a graph and returns the edges of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

Yields

generator
Yields a generator of Path objects, each of which is given as a list of the paths' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_path_nodes

```
enumerate_path_nodes(source=None, sink=None, minlength=1,
maxlength=None, edge_weight=None, node_weight=None, minedgewt=None,
maxedgewt=None, minnodewt=None, maxnodewt=None, maxtime=None)
```

Enumerates the paths in a graph and returns the nodes of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

Yields

generator
Yields a generator of Path objects, each of which is given as a list of the paths' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_paths

```
enumerate_paths(source=None, sink=None, minlength=1, maxlength=None,
edge_weight=None, node_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, *, data=True)
```

Enumerates the paths in a graph and returns the Path graph objects.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one Path object. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_sequence_shortest_paths

```
enumerate_sequence_shortest_paths(sequence, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None, *, data=True)
```

Enumerates the shortest paths in a graph that visit the indicated sequence nodes in order.

Parameters

sequence : iterable of nodes
Specifies the order in which to visit the required nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Path objects; each object represents a subpath between a sequence of nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_path_edges

```
enumerate_shortest_path_edges(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None)
```

Enumerates the shortest paths in a graph and returns the edges of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

Yields

generator
Yields a generator of paths; each path is given as a list of the paths' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_path_nodes

```
enumerate_shortest_path_nodes(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None)
```

Enumerates the shortest paths in a graph and returns the nodes of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

Yields

generator
Yields a generator of paths; each path is given as a list of the paths' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_paths

```
enumerate_shortest_paths(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None, *, data=True)
```

Enumerates the shortest paths in a graph and returns the Path graph objects.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Path objects; each object represents one shortest path. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

get_meta_graph

```
get_meta_graph(node_type, edge_type=None, immutable=None)
```

Constructs a metagraph from the input graph.

Parameters

node_type : str
Specifies the node attribute to consider in constructing the metagraph.

edge_type : str, optional
Specifies the edge attribute to consider in constructing the metagraph. By default, all connections between nodes are considered the same.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

Returns

SAS graph object
Returns the metagraph, which has an edge attribute count and determines the number of unique connections between pairs of nodes.

get_nlargest_biconcomps

```
get_nlargest_biconcomps(n, order_by='node', *, data=True, ascending=False)
```

Returns a list of up to *n* largest biconnected components.

Parameters

n : int
Specifies the number of graphs to return.

order_by : 'node' or 'edge' or None
Specifies whether to sort the biconnected components by the number of nodes or edges within those components. The default is None, which means that the biconnected components are not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. When the value is True, the smallest biconnected components are returned instead. The default is False.

Returns

list
Returns a list of n largest or smallest biconnected components of a graph.

get_nlargest_concomps

```
get_nlargest_concomps(n, order_by='node', *, strongly=True, data=True,
ascending=False)
```

Returns a list of up to *n* largest connected (with highest number of nodes) components of a graph.

Parameters

n : int
Specifies the number of graphs to return.

order_by : 'node' or 'edge'
Specifies whether to sort the component by number of nodes or edges within that component. The default is None, which means that the result is not sorted.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Returns

list
Returns a list of the n largest connected components of a graph.

has_edge_attribute

```
has_edge_attribute(name, *, all_edges=False)
```

Checks the edges of the input graph for the specified edge attribute.

Parameters

name : str
Specifies the name of the edge attribute.

all_edges : bool, optional
Specifies whether or not to check for the existence of the edge attribute in all or any of the edges. The default is False.

Returns

bool
If the all_edges parameter value is True, the value True is returned if that edge attribute exists in all edges; otherwise the value False is returned. If the all_edges parameter value is False, the value True is returned if at least one edge in the graph has that specified edge attribute.

has_node_attribute

```
has_node_attribute(name, *, all_nodes=False)
```

Checks the nodes of the input graph for the specified node attribute.

Parameters

name : str
Specifies the name of the node attribute.

all_nodes : bool, optional
Specifies whether or not to check for the existence of the node attribute in all or any of the nodes. The default is False.

Returns

bool
If the all_nodes parameter value is True, the value True is returned if that node attribute exists in all node; otherwise the value False is returned. If the all_nodes parameter value is False, the value True is returned if at least one node in the graph has that specified node attribute.

intersection

```
intersection(rhs, nodes_attrs_agg=None, edges_attrs_agg=None)
```

Returns the intersection of input graphs by finding the intersections of their nodes and edges.

The node attributes and edge attributes are aggregated using the rules that are defined by the *nodes_attrs_agg* and *edges_attrs_agg* parameters.

Parameters

rhs : SAS graph object
Specifies the other graph, of the same type as *self*, with which to intersect.

nodes_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate node attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps node attributes to functions. The default is None.

edges_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate edge attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps edge attributes to functions. The default is None.

Returns

SAS graph object
Returns a graph that is constructed by the intersection of the two graphs.

is_biconnected

```
is_biconnected()
```

Specifies whether or not the input graph is biconnected.

Returns

bool
Indicates whether the input graph is biconnected (True) or not (False).

is_bipartite

```
is_bipartite()
```

Determines whether the input graph is bipartite.

Returns

bool
Indicates whether the input graph is bipartite.

is_connected

```
is_connected()
```

Specifies whether or not the input graph is connected.

Returns

bool
Indicates whether the input graph is connected (True) or not (False).

is_cycle

```
is_cycle()
```

Determines whether the input graph is a simple cycle.

Returns

bool
Indicates whether the input graph is a simple cycle (True) or not (False).

is_dag

```
is_dag()
```

Determines whether the input graph is a directed acyclic graph (DAG).

Returns

bool
Indicates whether the input graph is a directed acyclic graph (DAG).

is_directed

```
is_directed()
```

Specifies whether or not the graph is directed.

Returns

bool
Indicates whether or not the graph is directed.

is_hamiltonian

```
is_hamiltonian(maxtime=None)
```

Determines whether the graph is Hamiltonian (has a Hamiltonian cycle) or not.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time to determine whether or not the graph is Hamiltonian. The default is None.

Returns

bool or None
Indicates whether or not the input graph is determined to be Hamiltonian. Returns None if no determination is made.

is_immutable

```
is_immutable()
```

Specifies whether or not the graph is immutable.

Returns

bool
Indicates whether or not the graph is immutable.

is_multigraph

```
is_multigraph()
```

Specifies whether or not the graph allows multiedges.

Returns

bool
Indicates whether or not the graph allows multiedges.

is_path

```
is_path()
```

Determines whether the input graph is a simple path.

Returns

bool
Indicates whether the input graph is a simple path (True) or not (False).

is_strongly_connected

```
is_strongly_connected()
```

Specifies whether the input graph is strongly connected.

Returns

bool
Indicates whether the input graph is strongly connected (True) or not (False).

is_weakly_connected

```
is_weakly_connected()
```

Specifies whether the input graph is weakly connected.

Returns

bool
Indicates whether the input graph is weakly connected (True) or not (False).

label_propagation

```
label_propagation(edge_weight=None, maxiters=100, tolerance=0.05, *,
data=True)
```

Computes the communities of the graph by using the label propagation algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of label changes for all nodes in the graph is less than the value. The default is 0.05.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one community. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

label_propagation_nodes

```
label_propagation_nodes(edge_weight=None, maxiters=100, tolerance=0.05, *,
data=True, inplace=False)
```

Computes community assignments for all nodes in the graph by using the label propagation algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of label changes for all nodes in the graph is less than the value. The default is 0.05.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

inplace : bool, optional
Specifies whether to add the community values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains the node attribute *community*, which denotes the community assignment for each node. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

leaf_node_count

```
leaf_node_count()
```

Computes a graph's leaf node count.

Returns

int
Returns the number of leaf nodes.

local_transitivity

```
local_transitivity(*, inplace=False)
```

Computes the local transitivity (clustering coefficient) and stores the results as node attributes on the graph.

Parameters

inplace : bool, optional
Specifies whether to add the transitivity values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains node attribute 'transitivity'. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

louvain

```
louvain(*, edge_weight=None, resolution=1, tolerance=0.001, maxiters=100,
data=True)
```

Computes the communities of the graph by using the Louvain algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

resolution : float, optional
Specifies the factor in the modularity equation that affects the expected number of links between two nodes. A higher value produces more communities. The default is 1.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of modularity gain between two consecutive iterations is less than the value. The default is 0.001.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one community. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

louvain_nodes

```
louvain_nodes(*, edge_weight=None, resolution=1, tolerance=0.001,
maxiters=100, data=True, inplace=False)
```

Computes community assignments for all nodes in the graph by using the Louvain algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

resolution : float, optional
Specifies the factor in the modularity equation that affects the expected number of links between two nodes. A higher value produces more communities. The default is 1.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of modularity gain between two consecutive iterations is less than the value. The default is 0.001.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

inplace : bool, optional
Specifies whether to add the community values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains the node attribute *community*, which denotes the community assignment for each node. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

maxflow

```
maxflow(source, sink, capacity_attr='upper', *, inplace=False, data=True)
```

Calculates the maximum flow from a source node to a sink node in a graph.

Parameters

source : number or str
Specifies the source node for the maximum flow.

sink : number or str
Specifies the sink node for the maximum flow.

capacity_attr : str, optional
Specifies the name of the edge attribute that contains the capacity of the edge. The default is 'upper'.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute *flow*, which contains the flow values. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, the returned graph contains only the flow edges.

maximal_cliques

```
maximal_cliques(maxcliques='ALL', node_weight=None, edge_weight=None,
minsize=1, maxsize=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, order_by=None, *,
data=True, ascending=False)
```

Enumerates all maximal cliques of the graph.

Parameters

maxcliques : int or 'ALL', optional
Specifies the maximum number of enumerated cliques to return. The default is 'ALL'.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

minsize : int, optional
Specifies the minimum number of nodes in a clique. The default is 1.

maxsize : int or None, optional
Specifies the maximum number of nodes in a clique. The default is None.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a clique. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a clique. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a clique. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a clique. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds for enumeration to spend. The default is None.

order_by : 'node' or None, optional
Specifies whether to sort the cliques by the number of nodes within each clique. The default is None, which means that the result is not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one maximal clique. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

min_cut

```
min_cut(edge_weight=None, *, inplace=False, data=True)
```

Calculates a minimum cut of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted minimum cut is calculated.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes

of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute 'mincut', which indicates the edges to include in the minimum cut set, and the node attribute 'mincut_partition', which indicates the node partition of the calculated minimum cut. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, a modified copy of the graph is returned.

min_s_t_cut

```
min_s_t_cut(source, sink, edge_weight=None, *, inplace=False, data=True)
```

Calculates a minimum s-t cut of a graph.

Parameters

source : number or str
Specifies the source node (s) for the minimum s-t cut.

sink : number or str
Specifies the sink node (t) for the minimum s-t cut.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted minimum s-t cut is calculated.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute *s_t_cut*, which indicates the edges to include in the minimum s-t cut set; and the node attribute *s_t_cut_partition*, which indicates the node partition of the calculated minimum s-t cut. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is

modified in place and None is returned. When the *inplace* parameter value is False, a modified copy of the graph is returned.

mincostflow

```
mincostflow(demand_lower_attr='lower', demand_upper_attr='upper',
capacity_lower_attr='lower', capacity_upper_attr='upper', cost_attr='weight',
maxtime=None, *, inplace=False, data=True)
```

Calculates the minimum-cost network flow for the graph.

Parameters

demand_lower_attr : str, optional
Specifies the name of the node attribute that contains the lower bound of the node supply. The default is 'lower'.

demand_upper_attr : str, optional
Specifies the name of the node attribute that contains the upper bound of the node supply. The default is 'upper'.

capacity_lower_attr : str, optional
Specifies the name of the edge attribute that contains the capacity lower bound of the edge. The default is 'lower'.

capacity_upper_attr : str, optional
Specifies the name of the edge attribute that contains the capacity upper bound of the edge. The default is 'upper'.

cost_attr : str, optional
Specifies the name of the edge attribute that contains the edge cost. The default is 'weight'.

maxtime : float or None, optional
Specifies the maximum amount of time to allow for computing the minimum-cost network flow. The default is None.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attributes *flow* and *reduced_cost*, which contain the flow and reduced cost values, respectively; and the node attribute *dual*, which contains the optimal dual value. This graph has the

solution_summary attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned.

minimum_spanning_tree

```
minimum_spanning_tree(edge_weight=None, source=None, *, inplace=False,
data=True)
```

Calculates a minimum spanning tree (MST) or forest of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

source : number or str or None
Specifies the source node for a directed MST. The default is None.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute 'mst', which indicates the edges to include in the spanning tree or forest. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, the returned graph contains only the spanning tree or forest edges.

minimum_weight_full_matching

```
minimum_weight_full_matching(edge_weight=None, *, data=True)
```

Returns a minimum-weight full matching of the bipartite graph *self*.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns a directed graph whose edges identify the assignment mapping in the minimum-weight full matching solution. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

neighbors

```
neighbors(n)
```

Returns an iterator over all neighbors of node n.

Parameters

n : node
Specifies a node in the graph.

Returns

iterator
Returns an iterator over all neighbors of node n.

new_edge_key

```
new_edge_key(u, v)
```

Returns an unused key for the edges between nodes *u* and *v*.

Parameters

u : numeric or str
Specifies the *from* node to add to the graph.

v : numeric or str
Specifies the *to* node to add to the graph.

Returns

int
Returns a unique edge key value.

node_clique_numbers

```
node_clique_numbers(*, inplace=False, data=True)
```

Computes node clique numbers and stores the results as attributes on the graph.

Parameters

inplace : bool, optional
Specifies whether to add the clique number values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the value of the *inplace* parameter is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph or modifies the existing graph to contain the node attribute *clique_number*, which includes the clique number of each node. The graph g has the attribute g.solution_summary, which contains information about the results of the algorithm.

number_of_edges

```
number_of_edges(u=None, v=None)
```

Returns the number of edges in the graph.

When you specify *u* and *v*, the number of edges between *u* and *v* is returned. When you specify *u* but *v* is None, the number of edges that connect to *u* is returned. When you specify *v* but *u* is None, the number of edges that connect to *v* is returned.

Parameters

u : numeric or str or None, optional
Specifies the *from* node. The default is None.

v : numeric or str or None, optional
Specifies the *to* node. The default is None.

Returns

int
Returns the number of edges in the graph. When you specify *u* and *v*, the number of edges between *u* and *v* is returned. When you specify *u* but *v* is None, the number of edges that connect to *u* is returned. When you specify *v* but *u* is None, the number of edges that connect to *v* is returned.

number_of_nodes

```
number_of_nodes()
```

Returns the number of nodes in the graph.

Returns

int
Returns the number of nodes in the graph.

projected_graph

```
projected_graph(nodes, neighbors=None, directed_method=None,
_measure=None, *, multigraph=False, data=True)
```

Computes a projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in self other than *nodes*.

directed_method : {‘converging’, ‘diverging’, ‘transitive’}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used. When the *directed_method* parameter is specified for an undirected graph, a ValueError is raised.

multigraph : bool, optional
Specifies whether or not to return a multigraph. When the value is True, the algorithm returns a multigraph in which the multiple edges represent multiple shared neighbors. The edge attribute ‘neighbor’ stores the label of the shared neighbor. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

projected_graph_adamic_adar

```
projected_graph_adamic_adar(nodes, neighbors=None, directed_method=None,
*, data=True)
```

Computes an Adamic-Adar projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than *nodes*.

directed_method : {‘converging’, ‘diverging’, ‘transitive’}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'adamic_adar' edge attribute.

projected_graph_common_neighbors

```
projected_graph_common_neighbors(nodes, neighbors=None,
directed_method=None, *, data=True)
```

Computes a common neighbors projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'common_neighbors' edge attribute.

projected_graph_cosine

```
projected_graph_cosine(nodes, edge_weight=None, neighbors=None,
directed_method=None, *, data=True)
```

Computes a cosine projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'cosine' edge attribute.

projected_graph_dice

```
projected_graph_dice(nodes, neighbors=None, directed_method=None, *,
data=True)
```

Computes a Sørensen-Dice projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the subset of nodes that are specified by the *neighbors* parameter. The strength of association in the calculated projection is represented by the 'dice' edge attribute.

projected_graph_jaccard

```
projected_graph_jaccard(nodes, neighbors=None, directed_method=None, *,
data=True)
```

Computes a Jaccard projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'jaccard' edge attribute.

query

```
query(q, expr=None, _node_attr_to_ignore=None, _edge_attr_to_ignore=None, *,
induced=False, break_symmetry=False, _generate_output=True,
_orbits_only=False, **kwargs)
```

Queries for the subgraph isomorphisms of q within g.

Parameters

q : graph
Specifies the query graph.

expr : _Expression or None, optional
Specifies additional logic to filter the query results. The default is None.

induced : bool, optional
Specifies whether or not to return only node-induced matches. The default is False.

break_symmetry : bool, optional
Specifies whether or not to break symmetry and eliminate automorphic relationships between matches. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one isomorphic mapping from *q* to *self*. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

reach_graph

```
reach_graph(source, max_reach=1, *, data=True)
```

Computes the reach graph for the source or set of sources.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to compute the reach network.

max_reach : int, optional
Specifies the number of steps (or hops) to traverse from the source nodes. The default is 1.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns a graph that represents the reach network. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

reach_graph_per_source

```
reach_graph_per_source(source, max_reach=1, *, data=True)
```

Computes a separate reach graph for each given source node.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to compute the reach network.

max_reach : int, optional
Specifies the number of steps (or hops) to traverse from the source nodes. The default is 1.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one reach network. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

remove_edge

```
remove_edge(u, v, key=None)
```

Removes a single edge from the graph.

Parameters

u : numeric or str
Specifies the *from* node.

v : numeric or str
Specifies the *to* node.

key : numeric or str or None
Specifies the edge key. The default is None, which means that when there are multiple edges between u and v, the most recently added edge is removed.

Examples

```
G.remove_edge(1,2)
```

remove_edges_from

```
remove_edges_from(edges)
```

Removes edges from the graph.

Parameters

edges : an iterable container of edges
When you specify two-tuples (u, v), the most recently added edge between u and v is removed. When you specify three-tuples (u, v, key) or four-tuples (u, v, key, data), the edge between u and v with the specified key is removed, where data are ignored.

remove_isolated_nodes

```
remove_isolated_nodes(immutable=None, *, inplace=False)
```

Removes isolated nodes from a graph.

Parameters

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the input graph. The default is False.

Returns

SAS graph object or None
Returns the resulting graph after isolated nodes are removed, or returns nothing if the *inplace* parameter value is True.

remove_node

```
remove_node(node)
```

Removes a single node from the graph.

Parameters

node : numeric or str
Specifies the node to remove from the graph.

remove_nodes_from

```
remove_nodes_from(nodes)
```

Removes nodes from the graph. If a node does not exist in the graph, it is silently ignored.

Parameters

nodes : an iterable container of nodes
Specifies the nodes to remove from the graph.

set_edge_attributes

```
set_edge_attributes(values, name=None, immutable=None, *, inplace=False)
```

Sets edge attributes by using a given value or dictionary of values.

Parameters

values : scalar or dict-like or Pandas DataFrame
Specifies what the edge attribute should be set to. When this parameter is set to a Pandas DataFrame, it is expected to have from, to, and edge_key (only for multigraphs) columns, and the other columns are treated as edge attributes to add. When *values* is a dictionary, the keys are edges and the values are either edge attribute values or dictionaries of attribute name-value pairs. For multigraphs, the edge tuples must be of the form (u, v, key). For simple graphs, the keys must be tuples of the form (u, v). When this parameter is not set to a

dictionary, it is treated as a single attribute value that is then applied to every edge in *self*. The attribute name is specified by the *name* parameter.

name : str or None, optional
Specifies the name of the edge attribute to set if the *values* parameter is set to a scalar. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is False, a copy of the input graph with the new edge attributes is returned. If the *inplace* parameter value is True, the graph is modified by setting the specified edge attributes.

set_node_attributes

```
set_node_attributes(values, name=None, immutable=None, *, inplace=False)
```

Sets node attributes from a given value or dictionary of values.

Parameters

values : scalar or dict-like or Pandas DataFrame
Specifies what the node attribute should be set to. When *values* is not a dictionary, it is treated as a single attribute value that is then applied to every node in *self*. This means that if you provide a mutable object, such as a list, updates to that object are reflected in the node attribute for every node. The attribute name is specified by the *name* parameter. When *values* is a dictionary, the keys are nodes and the values are either node attribute values or dictionaries of attribute name-value pairs.

name : str or None, optional
Specifies the name of the node attribute to set if the *values* parameter is set to a scalar. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not the graph is modified in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by setting the specified node attributes. If the *inplace* parameter value is False, a copy of the input graph with the new node attributes is returned.

shortest_path_distances

```
shortest_path_distances(source=None, sink=None, edge_weight=None,
minedgewt=None, maxedgewt=None)
```

Calculates the shortest path distances in a graph.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) and restricts enumeration to the shortest paths that contain the specified source node(s). The default is None.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) and restricts enumeration to the shortest paths that contain the specified sink node(s). The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

Returns

dict
Returns a dictionary where each key is a (source, sink) tuple and its value is the corresponding shortest source-sink path distance.

similarity_adamic_adar

```
similarity_adamic_adar(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Adamic-Adar node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_common_neighbors

```
similarity_common_neighbors(source, sink=None, top_k=None, bottom_k=None,
*, exclude_source=False, exclude_existing_neighbors=False)
```

Computes common neighbors node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_cosine

```
similarity_cosine(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes cosine node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_dice

```
similarity_dice(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Sørensen-Dice node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_graph

```
similarity_graph(measure, min_score=2.220446049250313e-16, max_score=inf,
*, exclude_source=False, exclude_existing_neighbors=False, data=True,
**kwargs)
```

Computes a node similarity graph.

Parameters

measure : str, one of {'adamic_adar', 'common_neighbors', 'cosine', 'dice', 'jaccard'}
Specifies the metric to be used to compute the node similarity.

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the edge attribute that corresponds to the measure name. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_adamic_adar

```
similarity_graph_adamic_adar(min_score=2.220446049250313e-16,
max_score=inf, *, exclude_source=False, exclude_existing_neighbors=False,
data=True)
```

Computes an Adamic-Adar node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'adamic_adar' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_common_neighbors

```
similarity_graph_common_neighbors(min_score=2.220446049250313e-16,
max_score=inf, *, exclude_source=False, exclude_existing_neighbors=False,
data=True)
```

Computes a common neighbors node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'common_neighbors' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_cosine

```
similarity_graph_cosine(edge_weight=None,
min_score=2.220446049250313e-16, max_score=inf, *, exclude_source=False,
exclude_existing_neighbors=False, data=True)
```

Computes a cosine node similarity graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'cosine' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_dice

```
similarity_graph_dice(min_score=2.220446049250313e-16, max_score=inf, *,
exclude_source=False, exclude_existing_neighbors=False, data=True)
```

Computes a Sørensen-Dice node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the *dice* edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_jaccard

```
similarity_graph_jaccard(min_score=2.220446049250313e-16, max_score=inf, *,
exclude_source=False, exclude_existing_neighbors=False, data=True)
```

Computes a Jaccard node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'jaccard' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_jaccard

```
similarity_jaccard(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Jaccard node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

singleton_node_count

```
singleton_node_count()
```

Computes a graph's singleton node count.

Returns

int
Returns the number of singleton nodes.

to_directed

```
to_directed()
```

Returns a directed copy of the input graph.

Returns

SAS graph object
Returns a directed copy of the input graph.

to_immutable

```
to_immutable()
```

Creates an immutable graph from the input graph.

Returns

SAS graph object
Returns an immutable SAS graph object.

to_mutable

```
to_mutable()
```

Creates a mutable graph from the input graph.

Returns

SAS graph object
Returns a mutable SAS graph object.

to_pandas_edges

```
to_pandas_edges(*, data=True, _for_conversion_to_immutable=False)
```

Returns the graph edges as a Pandas DataFrame.

Parameters

data : bool or list, optional
Specifies whether or not to retain attributes. The default is True. When the value is True (or when a list is given), the Pandas DataFrame includes all edge attributes (or all listed edge attributes).

Returns

dataframe
Returns a Pandas DataFrame that includes the graph edges.

to_pandas_nodes

```
to_pandas_nodes(*, data=True, _for_conversion_to_immutable=False)
```

Returns the graph nodes as a Pandas DataFrame.

Parameters

data : bool or list, optional
Specifies whether or not to retain attributes. The default is True. When the value is True (or when a list is given), the Pandas DataFrame includes all the node attributes (or all the listed attributes).

Returns

dataframe
Returns a Pandas DataFrame that includes the graph nodes.

to_undirected

```
to_undirected()
```

Returns a directed copy of the input graph.

Returns

SAS graph object
Returns a directed copy of the input graph.

topological_ordering

```
topological_ordering()
```

Finds a topological ordering of the graph's nodes.

Yields

generator
Yields a generator of nodes in topological order. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

transitive_closure

```
transitive_closure()
```

Calculates the transitive closure of a graph.

Returns

SAS graph object
Returns the transitive closure of the input graph that is returned as a SAS MultiGraph or MultiDiGraph object. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

triangle_count

```
triangle_count()
```

Computes a graph's triangle count.

Returns

int
Returns the number of triangles.

tsp_tour

```
tsp_tour(edge_weight=None, maxtime=None, minobjective=None, *,
inplace=False, use_milp=True, data=True, **options)
```

Calculates a traveling salesman problem (TSP) tour of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted TSP is calculated.

maxtime : float or None, optional
Specifies the maximum amount of time for solving a TSP. The default is None.

minobjective : float, optional
Specifies a minimum value such that when a solution is found whose objective is less than or equal to this value, the algorithm stops.

inplace : bool, optional
Specifies whether to add the TSP order values as attributes to the input graph directly or to create a copy of the graph. The default is False.

use_milp : bool, optional
Specifies whether or not to use a mixed integer linear programming (MILP) solver to solve a TSP. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is False, the returned graph is a Cycle that contains the calculated TSP tour edges. When the *inplace* parameter value is True, the graph is modified in place and None is returned. The modified graph contains the edge attributes *tsp* and *tsp_order* and the node attribute *tsp_order*.

union

```
union(rhs, nodes_attrs_agg=None, edges_attrs_agg=None)
```

Returns the union of input graphs by finding the unions of their nodes and edges.

The node attributes and edge attributes are aggregated using the rules that are defined by the *nodes_attrs_agg* and *edges_attrs_agg* parameters.

Parameters

rhs : SAS graph object
Specifies the other graph, of the same type as *self*, with which to find the union.

nodes_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate node attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps node attributes to functions. The default is None.

edges_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate edge attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps edge attributes to functions. The default is None.

Returns

SAS graph object
Returns a graph that is constructed by the union of the two graphs.

vrp

```
vrp(capacity, depot, node_demand='demand', edge_cost='weight', minroutes=1,
maxroutes=None, maxtime=None, minobjective=None, *, use_milp=True,
data=True, **options)
```

Solves the vehicle routing problem.

Parameters

capacity : float
Specifies the capacity of each vehicle.

depot : node
Specifies the depot node for the vehicle routing problem.

node_demand : str or dict or iterable
Specifies the quantity of goods to deliver to or pick up from a customer. The default is 'demand'.

edge_cost : str or dict or iterable
Specifies the cost of traversing each edge. The default is 'weight'.

minroutes : int, optional
Specifies the minimum number of routes that are allowed to service demand.

maxroutes : int, optional
Specifies the maximum number of routes that are allowed to service demand.

maxtime : float or None, optional
Specifies the maximum amount of time for solving the vehicle routing problem. The default is None.

minobjective : float, optional
Specifies a minimum value such that when a solution is found whose objective is less than or equal to this value, the algorithm stops.

use_milp : bool, optional
Specifies whether or not to use a mixed integer linear programming (MILP) solver to solve the vehicle routing problem. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Cycle objects; each cycle represents one route. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

vrptw

```
vrptw(capacity, depot, node_demand='demand', edge_cost='weight',
travel_time='weight', timewindow_lower=None, timewindow_upper=None,
service_time=None, minroutes=1, maxroutes=None, maxtime=None, *,
data=True)
```

Solves the vehicle routing problem with time windows.

Parameters

capacity : float
Specifies the capacity of each vehicle.

depot : node
Specifies the depot node for the vehicle routing problem.

node_demand : str or dict or iterable
Specifies the quantity of goods to deliver to or pick up from a customer. The default is 'demand'.

edge_cost : str or dict or iterable
Specifies the cost of traversing each edge.

travel_time : str or dict or iterable
Specifies attributes or values that define the time that it takes to traverse each edge. The default is 'weight'.

timewindow_lower : str or dict or iterable or None, optional
Specifies attributes or values that define the lower time limits within which a delivery vehicle can arrive at the customer nodes. The default is None.

timewindow_upper : str or dict or iterable or None, optional
Specifies attributes or values that define the upper time limits within which a delivery vehicle can arrive at the customer nodes. The default is None.

service_time : str or dict or iterable or None, optional
Specifies attributes or values that define the service time of the nodes. The default is None.

minroutes : int, optional
Specifies the minimum number of routes that are allowed to service demand.

maxroutes : int, optional
Specifies the maximum number of routes that are allowed to service demand.

maxtime : float or None, optional
Specifies the maximum amount of time for solving the vehicle routing problem. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Cycle objects; each cycle represents one route. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

23

Immutable Graph Classes

Classes

Class Name	Description
ImmutableDiGraph	Base class for directed immutable graphs.
ImmutableGraph	Base class for undirected immutable graphs.
ImmutableMultiDiGraph	Base class for directed immutable multigraphs.
ImmutableMultiGraph	Base class for undirected immutable multigraphs.

ImmutableDiGraph

Base class for directed immutable graphs.

API: ImmutableDiGraph

class
sasviya.network.classes.immutabledigraph.ImmutableDiGraph(graph=None, **kwargs)

Self-loops and multiedges are not allowed.

Parameters

graph : sasnet.DiGraph
Specifies the graph to convert to an ImmutableDiGraph object.

Attributes

edges
Returns the edges of the graph.

nodes
Returns the nodes of the graph.

Methods

Method Name	Description
add_core_number_attr	Computes the core numbers and stores the results as node attributes on the graph.
add_topological_order_attr	Computes a topological ordering and stores the results as node attributes of the graph.
adjacency	Returns an iterator over (node, adjacency dict) tuples for all nodes.
allows_multiedges	Specifies whether or not the graph allows multiedges.
allows_selfloops	Specifies whether or not the graph allows self-loops.
approximate_diameter	Computes an approximation of the graph's diameter.
articulation_point_count	Computes a graph's articulation point count.

Method Name	Description
articulation_points	Finds the articulation points (cut points) of a graph.
assortativity_degree	Computes a graph's degree assortativity.
assortativity_nominal	Computes a graph's nominal assortativity.
assortativity_numeric	Computes a graph's numeric assortativity.
average_shortest_path	Computes a graph's average shortest path.
biconcomp_distribution_edges	Computes the distribution of the number of edges in each biconnected components by using *numpy.histogram*.
biconcomp_distribution_nodes	Computes the distribution of the number of nodes in each biconnected components by using *numpy.histogram*.
biconcomp_size_edges	Computes the number of edges within each biconnected component.
biconcomp_size_nodes	Computes the number of nodes within each biconnected component.
biconnected_component_count	Computes a graph's biconnected component count.
biconnected_components	Computes the graph's biconnected components for an undirected graph.
block_cut_tree	Computes the block-cut tree of a graph.
centrality_betweenness	Computes weighted or unweighted betweenness centrality and stores the results as attributes on the graph.
centrality_closeness	Computes weighted or unweighted closeness centrality and stores the results as attributes on the graph.
centrality_degree	Computes weighted or unweighted degree centrality and stores the results as attributes on the graph.
centrality_eigenvector	Computes weighted or unweighted eigenvector centrality and stores the results as attributes on the graph.
centrality_hub_authority	Computes weighted or unweighted hub and authority centrality and stores the results as attributes on the graph.

Method Name	Description
centrality_influence	Computes weighted or unweighted influence centrality and stores the results as attributes on the graph.
centrality_pagerank	Computes weighted or unweighted PageRank centrality and stores the results as attributes on the graph.
concomp_distribution_edges	Computes the distribution of the number of edges in each connected component by using *numpy.histogram*.
concomp_distribution_nodes	Computes the distribution of the number of nodes in each connected component by using *numpy.histogram*.
concomp_size_edges	Computes the number of edges within each connected component.
concomp_size_nodes	Computes the number of nodes within each connected component.
connected_component_count	Computes a graph's connected component count.
connected_components	Computes the connected components of a graph.
copy	Returns a copy of the input graph.
core_decomposition	Computes a core decomposition of the graph.
cycle_count	Counts the number of cycles in a graph.
degree	Returns the degree of nodes.
density	Computes the density of the graph.
diameter	Computes a graph's diameter.
drop_edge_attributes	Removes edge attributes if they exist.
drop_node_attributes	Removes node attributes if they exist.
enumerate_cycle_edges	Enumerates the cycles in a graph and returns the edges of the cycles.
enumerate_cycle_nodes	Enumerates the cycles in a graph and returns the nodes of the cycles.
enumerate_cycles	Enumerates the cycles in a graph and returns the cycle graph objects.

Method Name	Description
enumerate_path_edges	Enumerates the paths in a graph and returns the edges of the paths.
enumerate_path_nodes	Enumerates the paths in a graph and returns the nodes of the paths.
enumerate_paths	Enumerates the paths in a graph and returns the Path graph objects.
enumerate_sequence_shortest_paths	Enumerates the shortest paths in a graph that visit the indicated sequence nodes in order.
enumerate_shortest_path_edges	Enumerates the shortest paths in a graph and returns the edges of the paths.
enumerate_shortest_path_nodes	Enumerates the shortest paths in a graph and returns the nodes of the paths.
enumerate_shortest_paths	Enumerates the shortest paths in a graph and returns the Path graph objects.
get_meta_graph	Constructs a metagraph from the input graph.
get_nlargest_biconcomps	Returns a list of up to *n* largest biconnected components.
get_nlargest_concomps	Returns a list of up to *n* largest connected (with highest number of nodes) components of a graph.
has_edge_attribute	Checks the edges of the input graph for the specified edge attribute.
has_node_attribute	Checks the nodes of the input graph for the specified node attribute.
in_degree	Returns the in-degree of nodes.
intersection	Returns the intersection of input graphs by finding the intersections of their nodes and edges.
is_biconnected	Specifies whether or not the input graph is biconnected.
is_bipartite	Determines whether the input graph is bipartite.
is_connected	Specifies whether or not the input graph is connected.
is_cycle	Determines whether the input graph is a simple cycle.

Method Name	Description
is_dag	Determines whether the input graph is a directed acyclic graph (DAG).
is_directed	Specifies whether or not the graph is directed.
is_hamiltonian	Determines whether the graph is Hamiltonian (has a Hamiltonian cycle) or not.
is_immutable	Specifies whether or not the graph is immutable.
is_multigraph	Specifies whether or not the graph allows multiedges.
is_path	Determines whether the input graph is a simple path.
is_strongly_connected	Specifies whether the input graph is strongly connected.
is_weakly_connected	Specifies whether the input graph is weakly connected.
label_propagation	Computes the communities of the graph by using the label propagation algorithm.
label_propagation_nodes	Computes community assignments for all nodes in the graph by using the label propagation algorithm.
leaf_node_count	Computes a graph's leaf node count.
local_transitivity	Computes the local transitivity (clustering coefficient) and stores the results as node attributes on the graph.
louvain	Computes the communities of the graph by using the Louvain algorithm.
louvain_nodes	Computes community assignments for all nodes in the graph by using the Louvain algorithm.
maxflow	Calculates the maximum flow from a source node to a sink node in a graph.
maximal_cliques	Enumerates all maximal cliques of the graph.
min_cut	Calculates a minimum cut of a graph.
min_s_t_cut	Calculates a minimum s-t cut of a graph.
mincostflow	Calculates the minimum-cost network flow for the graph.

Method Name	Description
minimum_spanning_tree	Calculates a minimum spanning tree (MST) or forest of a graph.
minimum_weight_full_matching	Returns a minimum-weight full matching of the bipartite graph *self*.
neighbors	Returns an iterator over all successor nodes of node n.
node_clique_numbers	Computes node clique numbers and stores the results as attributes on the graph.
number_of_edges	Returns the number of edges in the graph.
number_of_nodes	Returns the number of nodes in the graph.
out_degree	Returns the out-degree of nodes.
predecessors	Returns an iterator over all predecessor nodes of node n.
projected_graph	Computes a projected graph.
projected_graph_adamic_adar	Computes an Adamic-Adar projected graph.
projected_graph_common_neighbors	Computes a common neighbors projected graph.
projected_graph_cosine	Computes a cosine projected graph.
projected_graph_dice	Computes a Sørensen-Dice projected graph.
projected_graph_jaccard	Computes a Jaccard projected graph.
query	Queries for the subgraph isomorphisms of q within g.
reach_graph	Computes the reach graph for the source or set of sources.
reach_graph_per_source	Computes a separate reach graph for each given source node.
set_edge_attributes	Sets edge attributes by using a given value or dictionary of values.
set_node_attributes	Sets node attributes from a given value or dictionary of values.
shortest_path_distances	Calculates the shortest path distances in a graph.

Method Name	Description
similarity_adamic_adar	Computes Adamic-Adar node similarity.
similarity_common_neighbors	Computes common neighbors node similarity.
similarity_cosine	Computes cosine node similarity.
similarity_dice	Computes Sørensen-Dice node similarity.
similarity_graph	Computes a node similarity graph.
similarity_graph_adamic_adar	Computes an Adamic-Adar node similarity graph.
similarity_graph_common_neighbors	Computes a common neighbors node similarity graph.
similarity_graph_cosine	Computes a cosine node similarity graph.
similarity_graph_dice	Computes a Sørensen-Dice node similarity graph.
similarity_graph_jaccard	Computes a Jaccard node similarity graph.
similarity_jaccard	Computes Jaccard node similarity.
singleton_node_count	Computes a graph's singleton node count.
successors	Returns an iterator over all successor nodes of node n.
to_directed	Returns a directed copy of the input graph.
to_immutable	Creates an immutable graph from the input graph.
to_mutable	Creates a mutable graph from the input graph.
to_pandas_edges	Returns the graph edges as a Pandas DataFrame.
to_pandas_nodes	Returns the graph nodes as a Pandas DataFrame.
to_undirected	Returns a directed copy of the input graph.
topological_ordering	Finds a topological ordering of the graph's nodes.
transitive_closure	Calculates the transitive closure of a graph.
triangle_count	Computes a graph's triangle count.
tsp_tour	Calculates a traveling salesman problem (TSP) tour of a graph.

Method Name	Description
union	Returns the union of input graphs by finding the unions of their nodes and edges.
vrp	Solves the vehicle routing problem.
vrptw	Solves the vehicle routing problem with time windows.

add_core_number_attr

```
add_core_number_attr(maxtime=None, *, inplace=False, data=True)
```

Computes the core numbers and stores the results as node attributes on the graph.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time for core decomposition to spend. The default is None.

inplace : bool, optional
Specifies whether to add the core numbers as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph with core numbers that are stored as a node attribute named *core*. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

add_topological_order_attr

```
add_topological_order_attr(*, inplace=False, data=True)
```

Computes a topological ordering and stores the results as node attributes of the graph.

Parameters

inplace : bool, optional
Specifies whether or not to add the topological order values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a directed graph whose topological order is stored as a node attribute named *top_order*. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

adjacency

```
adjacency()
```

Returns an iterator over (node, adjacency dict) tuples for all nodes.

Returns

iterator
Returns an iterator of (node,adj dict) for all nodes.

Examples

```
G = sasnet.DiGraph([[1, 2, {'h': 2}],
                    [1, 3, {'w': 2, 'h': 0}],
                    [2, 3, {'w': 3, 'h': 5}]])
G_out = sasnet.ImmutableDiGraph(G)
print(list(G_out.adjacency()))
[(1, {2: {'h': 2}, 3: {'w': 2, 'h': 0}}), (2, {3: {'w': 3, 'h': 5}}), (3, {})].
```

allows_multiedges

```
allows_multiedges()
```

Specifies whether or not the graph allows multiedges.

Returns

bool
Indicates whether or not the graph allows multiedges.

allows_selfloops

```
allows_selfloops()
```

Specifies whether or not the graph allows self-loops.

Returns

bool
Indicates whether or not the graph allows self-loops.

approximate_diameter

```
approximate_diameter(edge_weight=None)
```

Computes an approximation of the graph's diameter.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted approximate diameter is computed.

Returns

float
Returns the approximate diameter.

articulation_point_count

```
articulation_point_count()
```

Computes a graph's articulation point count.

Returns

int
Returns the number of articulation points.

articulation_points

```
articulation_points()
```

Finds the articulation points (cut points) of a graph.

Returns

list
Returns a list of articulation points in the graph.

assortativity_degree

```
assortativity_degree(source_direction='out', target_direction='in',
edge_weight=None)
```

Computes a graph's degree assortativity.

Parameters

source_direction : {'in', 'out'}
Specifies the degree type (in-degree or out-degree) for the source node in a directed graph.

target_direction : {'in', 'out'}
Specifies the degree type (in-degree or out-degree) for the target node in a directed graph.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted assortativity degree is computed.

Returns

float
Returns the degree assortativity.

assortativity_nominal

```
assortativity_nominal(node_attr)
```

Computes a graph's nominal assortativity.

Parameters

node_attr : str
Specifies the name of the nominal node attribute to use.

Returns

float
Returns the nominal assortativity.

assortativity_numeric

```
assortativity_numeric(node_attr)
```

Computes a graph's numeric assortativity.

Parameters

node_attr : str
Specifies the name of the numeric node attribute to use.

Returns

float
Returns the numeric assortativity.

average_shortest_path

```
average_shortest_path(edge_weight=None)
```

Computes a graph's average shortest path.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted average shortest path is computed.

Returns

float
Returns the average shortest path.

biconcomp_distribution_edges

```
biconcomp_distribution_edges(bins=None)
```

Computes the distribution of the number of edges in each biconnected components by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is int (an integer), it specifies the number of equal-width bins. When *bins* is a list, it specifies bin ranges and should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is str (a string), it specifies the method to use for calculating the bin ranges. For example, 'stone','auto','doane','fd','rice','scott','sqrt','sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of biconnected components whose edge count is within the range of the bin.

biconcomp_distribution_nodes

```
biconcomp_distribution_nodes(bins=None)
```

Computes the distribution of the number of nodes in each biconnected components by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is int (an integer), it specifies the number of equal-width bins. When *bins* is a list, it specifies bin ranges and should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is str (a string), it specifies the method to use for calculating the bin ranges. For example, 'stone','auto','doane','fd','rice','scott','sqrt','sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of biconnected components whose node count is within the range of the bin.

biconcomp_size_edges

```
biconcomp_size_edges()
```

Computes the number of edges within each biconnected component.

Returns

dict
Returns a dictionary in which the keys are biconnected component IDs and the values are the number of edges within each component.

biconcomp_size_nodes

```
biconcomp_size_nodes()
```

Computes the number of nodes within each biconnected component.

Returns

dict
Returns a dictionary in which the keys are biconnected component IDs and the values are the number of nodes within each component.

biconnected_component_count

```
biconnected_component_count()
```

Computes a graph's biconnected component count.

Returns

int
Returns the number of biconnected components.

biconnected_components

```
biconnected_components(order_by=None, *, data=True, ascending=False)
```

Computes the graph's biconnected components for an undirected graph.

Parameters

order_by : 'node' or 'edge' or None, optional
Specifies the parameter that is used to sort the components by the number of nodes or edges within each component. The default is None, which means that the components are not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Returns the result as a generator of biconnected component subgraphs. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

block_cut_tree

```
block_cut_tree(*, return_nodes_map=False)
```

Computes the block-cut tree of a graph.

Parameters

return_nodes_map : bool, optional
Specifies whether or not to also compute the block-cut tree node map. The default is False.

Returns

SAS graph object
Returns a graph that represents the block-cut tree. When *return_nodes_map = True*, the output graph has the *nodes_map* attribute, which contains a dictionary that maps its nodes to a list of nodes in the original graph. Note: If a block is empty (that is, the corresponding biconnected component contains only articulation points), that block does not appear as a key in the *nodes_map* dictionary.

centrality_betweenness

```
centrality_betweenness(edge_weight=None, sample_percent=100, *,
inplace=False, data=True)
```

Computes weighted or unweighted betweenness centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that unweighted betweenness centrality is computed.

sample_percent : int, optional
Specifies the percentage of source nodes to sample for the approximate betweenness calculation. This parameter value should be in the range (0, 100]. The default is 100.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the node and edge attributes *between_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_closeness

```
centrality_closeness(edge_weight=None, nopath='diameter', *, inplace=False,
data=True)
```

Computes weighted or unweighted closeness centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted closeness centrality is computed.

nopath : {'diameter', 'harmonic', or 'zero'}, optional
Specifies a method to use for accounting for the shortest path distance between two nodes when a path does not exist. The valid values are as follows: - 'diameter' uses the graph diameter (plus one) as the shortest path distance between disconnected nodes. - 'harmonic' uses the harmonic formula for calculating closeness centrality. - 'zero' uses zero as the shortest path distance between disconnected nodes.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the closeness centrality node attributes *centr_close_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_degree

```
centrality_degree(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted degree centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted degree centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes

of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the degree centrality node attributes *centr_degree_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_eigenvector

```
centrality_eigenvector(edge_weight=None, maxiters=10000, *, inplace=False,
data=True)
```

Computes weighted or unweighted eigenvector centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted eigenvector centrality is computed.

maxiters : int, optional
Specifies the maximum number of iterations in order to limit the amount of computation time that is spent when convergence is slow. The default is 10,000.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the degree centrality node attributes *centr_eigen_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_hub_authority

```
centrality_hub_authority(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted hub and authority centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted hub and authority centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the hub and authority centrality node attributes *centr_hub_** and *centr_auth_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_influence

```
centrality_influence(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted influence centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted influence centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the first- and second-order influence centrality node attributes *centr_influence1_** and *centr_influence2_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_pagerank

```
centrality_pagerank(edge_weight=None, alpha=0.85, tolerance=1e-09, *,
inplace=False, data=True)
```

Computes weighted or unweighted PageRank centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted PageRank centrality is computed.

alpha : float, optional
Specifies the damping parameter for the PageRank centrality (or algorithm). The default is 0.85.

tolerance : float, optional
Specifies the tolerance parameter for the PageRank centrality (or algorithm). The default is 1E-9.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the PageRank centrality node attributes *centr_pagerank_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

concomp_distribution_edges

```
concomp_distribution_edges(bins=None, *, strongly=True)
```

Computes the distribution of the number of edges in each connected component by using *numpy.histogram*.

Parameters

bins : int or list or str, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is an integer (int), it represents the number of equal-width bins. When *bins* is a list, it specifies bin ranges and it should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is a string (str), it specifies the method to be used for calculating the bin ranges. For example, 'stone', 'auto', 'doane', 'fd', 'rice', 'scott', 'sqrt', 'sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of components whose edge count within the range of the bin.

concomp_distribution_nodes

```
concomp_distribution_nodes(bins=None, *, strongly=True)
```

Computes the distribution of the number of nodes in each connected component by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is an integer (int), it represents the number of equal-width bins. When *bins* is a list, it specifies bin ranges and it should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is a string (str), it specifies the method to use for calculating the bin ranges. For example, 'stone', 'auto', 'doane', 'fd', 'rice', 'scott', 'sqrt', 'sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of components whose node count is within the range of the bin.

concomp_size_edges

```
concomp_size_edges(*, strongly=True)
```

Computes the number of edges within each connected component.

Parameters

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary in which the keys are connected component IDs and the values are the number of edges within each component.

concomp_size_nodes

```
concomp_size_nodes(*, strongly=True)
```

Computes the number of nodes within each connected component.

Parameters

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary in which the keys are connected component IDs and the values are the number of nodes within each component.

connected_component_count

```
connected_component_count()
```

Computes a graph's connected component count.

Returns

int
Returns the number of connected components.

connected_components

```
connected_components(order_by=None, *, strongly=True, data=True,
ascending=False)
```

Computes the connected components of a graph.

Parameters

order_by : 'node' or 'edge' or None, optional
Specifies whether to sort the components by the number of nodes or edges within each component. The default is None, which means that the components are not sorted.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one connected component. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

copy

```
copy(immutable=None, *, data=True)
```

Returns a copy of the input graph.

Parameters

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, meaning that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns an independent copy of the input.

core_decomposition

```
core_decomposition(maxtime=None, *, data=True)
```

Computes a core decomposition of the graph.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time for core decomposition to spend. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one k-core in increasing core number. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

cycle_count

```
cycle_count(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Counts the number of cycles in a graph.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Returns

int
Returns a count of cycles in the graph.

degree

```
degree(n=None, edge_weight=None)
```

Returns the degree of nodes.

Parameters

n : single node or container of nodes or None, optional
Specifies which nodes to include. The default is None, which means that the degree of all nodes is returned.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None.

Returns

int or dictionary
Returns the degree of specified node(s).

density

```
density()
```

Computes the density of the graph.

Returns

float
Returns the density of the graph.

diameter

```
diameter(edge_weight=None)
```

Computes a graph's diameter.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted diameter is computed.

Returns

int or float
Returns the diameter.

drop_edge_attributes

```
drop_edge_attributes(attributes_subset=None, edges_subset=None,
immutable=None, *, inplace=False)
```

Removes edge attributes if they exist.

Parameters

attributes_subset : str or list, optional
Specifies the edge attribute or a list of the edge attributes to remove; for example, 'weight' or ['weight','color']. By default, all edge attributes are removed.

edges_subset : list or iterable of edges, optional
Specifies the edges to remove attributes from; for example, [(1,2),(2,3)]. By default the attributes for all edges are removed.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by removing the specified edge attributes. If the *inplace* parameter value is False, a copy of the input graph with the modified edge attributes is returned.

drop_node_attributes

```
drop_node_attributes(attributes_subset=None, nodes_subset=None,
immutable=None, *, inplace=False)
```

Removes node attributes if they exist.

Parameters

attributes_subset : str or list, optional
Specifies the node attribute or list of node attributes to remove; for example, ['weight','color']. By default, all node attributes are removed.

nodes_subset : list or iterable of nodes, optional
Specifies the nodes to drop attributes from; for example, [1,2,3] or (1,2,3) or [1] or (1,). By default, the attributes for all nodes are removed.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by removing the specified node attributes. If the *inplace* parameter value is False, a copy of the input graph with the modified node attributes is returned.

enumerate_cycle_edges

```
enumerate_cycle_edges(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Enumerates the cycles in a graph and returns the edges of the cycles.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Yields

generator
Yields a generator of cycles, each of which is given as a list of the cycles' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_cycle_nodes

```
enumerate_cycle_nodes(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Enumerates the cycles in a graph and returns the nodes of the cycles.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Yields

generator
Yields a generator of cycles, each of which is given as a list of the cycles' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_cycles

```
enumerate_cycles(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None, *,
data=True)
```

Enumerates the cycles in a graph and returns the cycle graph objects.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one cycle of the same type as the input graph. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_path_edges

```
enumerate_path_edges(source=None, sink=None, minlength=1,
maxlength=None, edge_weight=None, node_weight=None, minedgewt=None,
maxedgewt=None, minnodewt=None, maxnodewt=None, maxtime=None)
```

Enumerates the paths in a graph and returns the edges of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

Yields

generator
Yields a generator of Path objects, each of which is given as a list of the paths' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_path_nodes

```
enumerate_path_nodes(source=None, sink=None, minlength=1,
maxlength=None, edge_weight=None, node_weight=None, minedgewt=None,
maxedgewt=None, minnodewt=None, maxnodewt=None, maxtime=None)
```

Enumerates the paths in a graph and returns the nodes of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

Yields

generator
Yields a generator of Path objects, each of which is given as a list of the paths' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_paths

```
enumerate_paths(source=None, sink=None, minlength=1, maxlength=None,
edge_weight=None, node_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, *, data=True)
```

Enumerates the paths in a graph and returns the Path graph objects.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one Path object. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_sequence_shortest_paths

```
enumerate_sequence_shortest_paths(sequence, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None, *, data=True)
```

Enumerates the shortest paths in a graph that visit the indicated sequence nodes in order.

Parameters

sequence : iterable of nodes
Specifies the order in which to visit the required nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Path objects; each object represents a subpath between a sequence of nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_path_edges

```
enumerate_shortest_path_edges(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None)
```

Enumerates the shortest paths in a graph and returns the edges of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

Yields

generator
Yields a generator of paths; each path is given as a list of the paths' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_path_nodes

```
enumerate_shortest_path_nodes(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None)
```

Enumerates the shortest paths in a graph and returns the nodes of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

Yields

generator
Yields a generator of paths; each path is given as a list of the paths' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_paths

```
enumerate_shortest_paths(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None, *, data=True)
```

Enumerates the shortest paths in a graph and returns the Path graph objects.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Path objects; each object represents one shortest path. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

get_meta_graph

```
get_meta_graph(node_type, edge_type=None, immutable=None)
```

Constructs a metagraph from the input graph.

Parameters

node_type : str
Specifies the node attribute to consider in constructing the metagraph.

edge_type : str, optional
Specifies the edge attribute to consider in constructing the metagraph. By default, all connections between nodes are considered the same.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

Returns

SAS graph object
Returns the metagraph, which has an edge attribute count and determines the number of unique connections between pairs of nodes.

get_nlargest_biconcomps

```
get_nlargest_biconcomps(n, order_by='node', *, data=True, ascending=False)
```

Returns a list of up to *n* largest biconnected components.

Parameters

n : int
Specifies the number of graphs to return.

order_by : 'node' or 'edge' or None
Specifies whether to sort the biconnected components by the number of nodes or edges within those components. The default is None, which means that the biconnected components are not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. When the value is True, the smallest biconnected components are returned instead. The default is False.

Returns

list
Returns a list of n largest or smallest biconnected components of a graph.

get_nlargest_concomps

```
get_nlargest_concomps(n, order_by='node', *, strongly=True, data=True,
ascending=False)
```

Returns a list of up to *n* largest connected (with highest number of nodes) components of a graph.

Parameters

n : int
Specifies the number of graphs to return.

order_by : 'node' or 'edge'
Specifies whether to sort the component by number of nodes or edges within that component. The default is None, which means that the result is not sorted.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Returns

list
Returns a list of the n largest connected components of a graph.

has_edge_attribute

```
has_edge_attribute(name, *, all_edges=False)
```

Checks the edges of the input graph for the specified edge attribute.

Parameters

name : str
Specifies the name of the edge attribute.

all_edges : bool, optional
Specifies whether or not to check for the existence of the edge attribute in all or any of the edges. The default is False.

Returns

bool
If the all_edges parameter value is True, the value True is returned if that edge attribute exists in all edges; otherwise the value False is returned. If the all_edges parameter value is False, the value True is returned if at least one edge in the graph has that specified edge attribute.

has_node_attribute

```
has_node_attribute(name, *, all_nodes=False)
```

Checks the nodes of the input graph for the specified node attribute.

Parameters

name : str
Specifies the name of the node attribute.

all_nodes : bool, optional
Specifies whether or not to check for the existence of the node attribute in all or any of the nodes. The default is False.

Returns

bool
If the all_nodes parameter value is True, the value True is returned if that node attribute exists in all node; otherwise the value False is returned. If the

all_nodes parameter value is False, the value True is returned if at least one node in the graph has that specified node attribute.

in_degree

```
in_degree(n=None, edge_weight=None)
```

Returns the in-degree of nodes.

Parameters

n : single node or container of nodes or None, optional
Specifies which nodes to include. The default is None, which means that the in-degree of all nodes is returned.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None.

Returns

numeric or dictionary
Returns the in-degree of the node if a single node is passed, or returns a dictionary with nodes as keys and their in-degrees as values if None or a list of nodes is passed.

intersection

```
intersection(rhs, nodes_attrs_agg=None, edges_attrs_agg=None)
```

Returns the intersection of input graphs by finding the intersections of their nodes and edges.

The node attributes and edge attributes are aggregated using the rules that are defined by the *nodes_attrs_agg* and *edges_attrs_agg* parameters.

Parameters

rhs : SAS graph object
Specifies the other graph, of the same type as *self*, with which to intersect.

nodes_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate node attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps node attributes to functions. The default is None.

edges_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate edge attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps edge attributes to functions. The default is None.

Returns

SAS graph object
Returns a graph that is constructed by the intersection of the two graphs.

is_biconnected

```
is_biconnected()
```

Specifies whether or not the input graph is biconnected.

Returns

bool
Indicates whether the input graph is biconnected (True) or not (False).

is_bipartite

```
is_bipartite()
```

Determines whether the input graph is bipartite.

Returns

bool
Indicates whether the input graph is bipartite.

is_connected

```
is_connected()
```

Specifies whether or not the input graph is connected.

Returns

bool
Indicates whether the input graph is connected (True) or not (False).

is_cycle

```
is_cycle()
```

Determines whether the input graph is a simple cycle.

Returns

bool
Indicates whether the input graph is a simple cycle (True) or not (False).

is_dag

```
is_dag()
```

Determines whether the input graph is a directed acyclic graph (DAG).

Returns

bool
Indicates whether the input graph is a directed acyclic graph (DAG).

is_directed

```
is_directed()
```

Specifies whether or not the graph is directed.

Returns

bool
Indicates whether or not the graph is directed.

is_hamiltonian

```
is_hamiltonian(maxtime=None)
```

Determines whether the graph is Hamiltonian (has a Hamiltonian cycle) or not.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time to determine whether or not the graph is Hamiltonian. The default is None.

Returns

bool or None
Indicates whether or not the input graph is determined to be Hamiltonian. Returns None if no determination is made.

is_immutable

```
is_immutable()
```

Specifies whether or not the graph is immutable.

Returns

bool
Indicates whether or not the graph is immutable.

is_multigraph

```
is_multigraph()
```

Specifies whether or not the graph allows multiedges.

Returns

bool
Indicates whether or not the graph allows multiedges.

is_path

```
is_path()
```

Determines whether the input graph is a simple path.

Returns

bool
Indicates whether the input graph is a simple path (True) or not (False).

is_strongly_connected

```
is_strongly_connected()
```

Specifies whether the input graph is strongly connected.

Returns

bool
Indicates whether the input graph is strongly connected (True) or not (False).

is_weakly_connected

```
is_weakly_connected()
```

Specifies whether the input graph is weakly connected.

Returns

bool
Indicates whether the input graph is weakly connected (True) or not (False).

label_propagation

```
label_propagation(edge_weight=None, maxiters=100, tolerance=0.05, *,
data=True)
```

Computes the communities of the graph by using the label propagation algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of label changes for all nodes in the graph is less than the value. The default is 0.05.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one community. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

label_propagation_nodes

```
label_propagation_nodes(edge_weight=None, maxiters=100, tolerance=0.05, *,
data=True, inplace=False)
```

Computes community assignments for all nodes in the graph by using the label propagation algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of label changes for all nodes in the graph is less than the value. The default is 0.05.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

inplace : bool, optional
Specifies whether to add the community values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains the node attribute *community*, which denotes the community assignment for each node. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

leaf_node_count

```
leaf_node_count()
```

Computes a graph's leaf node count.

Returns

int
Returns the number of leaf nodes.

local_transitivity

```
local_transitivity(*, inplace=False)
```

Computes the local transitivity (clustering coefficient) and stores the results as node attributes on the graph.

Parameters

inplace : bool, optional
Specifies whether to add the transitivity values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains node attribute 'transitivity'. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

louvain

```
louvain(*, edge_weight=None, resolution=1, tolerance=0.001, maxiters=100,
data=True)
```

Computes the communities of the graph by using the Louvain algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

resolution : float, optional
Specifies the factor in the modularity equation that affects the expected number of links between two nodes. A higher value produces more communities. The default is 1.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of modularity gain between two consecutive iterations is less than the value. The default is 0.001.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one community. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

louvain_nodes

```
louvain_nodes(*, edge_weight=None, resolution=1, tolerance=0.001,
maxiters=100, data=True, inplace=False)
```

Computes community assignments for all nodes in the graph by using the Louvain algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

resolution : float, optional
Specifies the factor in the modularity equation that affects the expected number of links between two nodes. A higher value produces more communities. The default is 1.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of modularity gain between two consecutive iterations is less than the value. The default is 0.001.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

inplace : bool, optional
Specifies whether to add the community values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains the node attribute *community*, which denotes the community assignment for each node. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

maxflow

```
maxflow(source, sink, capacity_attr='upper', *, inplace=False, data=True)
```

Calculates the maximum flow from a source node to a sink node in a graph.

Parameters

source : number or str
Specifies the source node for the maximum flow.

sink : number or str
Specifies the sink node for the maximum flow.

capacity_attr : str, optional
Specifies the name of the edge attribute that contains the capacity of the edge. The default is 'upper'.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute *flow*, which contains the flow values. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, the returned graph contains only the flow edges.

maximal_cliques

```
maximal_cliques(maxcliques='ALL', node_weight=None, edge_weight=None,
minsize=1, maxsize=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, order_by=None, *,
data=True, ascending=False)
```

Enumerates all maximal cliques of the graph.

Parameters

maxcliques : int or 'ALL', optional
Specifies the maximum number of enumerated cliques to return. The default is 'ALL'.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

minsize : int, optional
Specifies the minimum number of nodes in a clique. The default is 1.

maxsize : int or None, optional
Specifies the maximum number of nodes in a clique. The default is None.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a clique. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a clique. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a clique. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a clique. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds for enumeration to spend. The default is None.

order_by : 'node' or None, optional
Specifies whether to sort the cliques by the number of nodes within each clique. The default is None, which means that the result is not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one maximal clique. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

min_cut

```
min_cut(edge_weight=None, *, inplace=False, data=True)
```

Calculates a minimum cut of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted minimum cut is calculated.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute 'mincut', which indicates the edges to include in the minimum cut set, and the node attribute 'mincut_partition', which indicates the node partition of the calculated minimum cut. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, a modified copy of the graph is returned.

min_s_t_cut

```
min_s_t_cut(source, sink, edge_weight=None, *, inplace=False, data=True)
```

Calculates a minimum s-t cut of a graph.

Parameters

source : number or str
Specifies the source node (s) for the minimum s-t cut.

sink : number or str
Specifies the sink node (t) for the minimum s-t cut.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted minimum s-t cut is calculated.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute *s_t_cut*, which indicates the edges to include in the minimum s-t cut set; and the node attribute *s_t_cut_partition*, which indicates the node partition of the calculated minimum s-t cut. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, a modified copy of the graph is returned.

mincostflow

```
mincostflow(demand_lower_attr='lower', demand_upper_attr='upper',
capacity_lower_attr='lower', capacity_upper_attr='upper', cost_attr='weight',
maxtime=None, *, inplace=False, data=True)
```

Calculates the minimum-cost network flow for the graph.

Parameters

demand_lower_attr : str, optional
Specifies the name of the node attribute that contains the lower bound of the node supply. The default is 'lower'.

demand_upper_attr : str, optional
Specifies the name of the node attribute that contains the upper bound of the node supply. The default is 'upper'.

capacity_lower_attr : str, optional
Specifies the name of the edge attribute that contains the capacity lower bound of the edge. The default is 'lower'.

capacity_upper_attr : str, optional
Specifies the name of the edge attribute that contains the capacity upper bound of the edge. The default is 'upper'.

cost_attr : str, optional
Specifies the name of the edge attribute that contains the edge cost. The default is 'weight'.

maxtime : float or None, optional
Specifies the maximum amount of time to allow for computing the minimum-cost network flow. The default is None.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attributes *flow* and *reduced_cost*, which contain the flow and reduced cost values, respectively; and the node attribute *dual*, which contains the optimal dual value. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned.

minimum_spanning_tree

```
minimum_spanning_tree(edge_weight=None, source=None, *, inplace=False,
data=True)
```

Calculates a minimum spanning tree (MST) or forest of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

source : number or str or None
Specifies the source node for a directed MST. The default is None.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute 'mst', which indicates the edges to include in the spanning tree or forest. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, the returned graph contains only the spanning tree or forest edges.

minimum_weight_full_matching

```
minimum_weight_full_matching(edge_weight=None, *, data=True)
```

Returns a minimum-weight full matching of the bipartite graph *self*.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns a directed graph whose edges identify the assignment mapping in the minimum-weight full matching solution. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

neighbors

```
neighbors(n)
```

Returns an iterator over all successor nodes of node n.

Parameters

n : node
Specifies a node in the graph.

Returns

iterator
Returns an iterator over all successor nodes of node n.

node_clique_numbers

```
node_clique_numbers(*, inplace=False, data=True)
```

Computes node clique numbers and stores the results as attributes on the graph.

Parameters

inplace : bool, optional
Specifies whether to add the clique number values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the value of the *inplace* parameter is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph or modifies the existing graph to contain the node attribute *clique_number*, which includes the clique number of each node. The graph g has the attribute g.solution_summary, which contains information about the results of the algorithm.

number_of_edges

```
number_of_edges(u=None, v=None)
```

Returns the number of edges in the graph.

When you specify *u* and *v*, the number of edges from *u* to *v* is returned. When you specify *u* but *v* is None, the number of connections from *u* to other nodes (out-degree of node *u*) is returned. When you specify *v* but *u* is None, the number of connections from other nodes to *v* (in-degree of node *v*) is returned.

Parameters

u : numeric or str or None, optional
Specifies the *from* node. The default is None.

v : numeric or str or None, optional
Specifies the *to* node. The default is None.

Returns

int
Returns the number of edges in the graph. When you specify *u* and *v*, the number of edges from *u* to *v* is returned. When you specify *u* but *v* is None, the number of connections from *u* to other nodes (out-degree of node *u*) is returned. When you specify *v* but *u* is None, the number of connections from other nodes to *v* (in-degree of node *v*) is returned.

number_of_nodes

```
number_of_nodes()
```

Returns the number of nodes in the graph.

Returns

int
Returns the number of nodes in the graph.

out_degree

```
out_degree(n=None, edge_weight=None)
```

Returns the out-degree of nodes.

Parameters

n : single node or container of nodes or None, optional
Specifies which nodes to include. The default is None, which means that the out-degree of all nodes is returned.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None.

Returns

numeric or dictionary
Returns the out-degree of the node if a single node is passed, or returns a dictionary with nodes as keys and their out-degrees as values if None or a list of nodes is passed.

predecessors

```
predecessors(n)
```

Returns an iterator over all predecessor nodes of node n.

Parameters

n : node
Specifies a node in the graph.

Returns

iterator
Returns an iterator over all predecessor nodes of node n.

projected_graph

```
projected_graph(nodes, neighbors=None, directed_method=None,
_measure=None, *, multigraph=False, data=True)
```

Computes a projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in self other than *nodes*.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used. When the *directed_method* parameter is specified for an undirected graph, a ValueError is raised.

multigraph : bool, optional
Specifies whether or not to return a multigraph. When the value is True, the algorithm returns a multigraph in which the multiple edges represent multiple shared neighbors. The edge attribute 'neighbor' stores the label of the shared neighbor. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

projected_graph_adamic_adar

```
projected_graph_adamic_adar(nodes, neighbors=None, directed_method=None,
*, data=True)
```

Computes an Adamic-Adar projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than *nodes*.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'adamic_adar' edge attribute.

projected_graph_common_neighbors

```
projected_graph_common_neighbors(nodes, neighbors=None,
directed_method=None, *, data=True)
```

Computes a common neighbors projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'common_neighbors' edge attribute.

projected_graph_cosine

```
projected_graph_cosine(nodes, edge_weight=None, neighbors=None,
directed_method=None, *, data=True)
```

Computes a cosine projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'cosine' edge attribute.

projected_graph_dice

```
projected_graph_dice(nodes, neighbors=None, directed_method=None, *,
data=True)
```

Computes a Sørensen-Dice projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the subset of nodes that are specified by the *neighbors* parameter. The strength of association in the calculated projection is represented by the 'dice' edge attribute.

projected_graph_jaccard

```
projected_graph_jaccard(nodes, neighbors=None, directed_method=None, *,
data=True)
```

Computes a Jaccard projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'jaccard' edge attribute.

query

```
query(q, expr=None, _node_attr_to_ignore=None, _edge_attr_to_ignore=None, *,
induced=False, break_symmetry=False, _generate_output=True,
_orbits_only=False, **kwargs)
```

Queries for the subgraph isomorphisms of q within g.

Parameters

q : graph
Specifies the query graph.

expr : _Expression or None, optional
Specifies additional logic to filter the query results. The default is None.

induced : bool, optional
Specifies whether or not to return only node-induced matches. The default is False.

break_symmetry : bool, optional
Specifies whether or not to break symmetry and eliminate automorphic relationships between matches. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one isomorphic mapping from *q* to *self*. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

reach_graph

```
reach_graph(source, max_reach=1, *, data=True)
```

Computes the reach graph for the source or set of sources.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to compute the reach network.

max_reach : int, optional
Specifies the number of steps (or hops) to traverse from the source nodes. The default is 1.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns a graph that represents the reach network. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

reach_graph_per_source

```
reach_graph_per_source(source, max_reach=1, *, data=True)
```

Computes a separate reach graph for each given source node.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to compute the reach network.

max_reach : int, optional
Specifies the number of steps (or hops) to traverse from the source nodes. The default is 1.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one reach network. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

set_edge_attributes

```
set_edge_attributes(values, name=None, immutable=None, *, inplace=False)
```

Sets edge attributes by using a given value or dictionary of values.

Parameters

values : scalar or dict-like or Pandas DataFrame
Specifies what the edge attribute should be set to. When this parameter is set to a Pandas DataFrame, it is expected to have from, to, and edge_key (only for multigraphs) columns, and the other columns are treated as edge attributes to add. When *values* is a dictionary, the keys are edges and the values are either edge attribute values or dictionaries of attribute name-value pairs. For multigraphs, the edge tuples must be of the form (u, v, key). For simple graphs, the keys must be tuples of the form (u, v). When this parameter is not set to a dictionary, it is treated as a single attribute value that is then applied to every edge in *self*. The attribute name is specified by the *name* parameter.

name : str or None, optional
Specifies the name of the edge attribute to set if the *values* parameter is set to a scalar. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is False, a copy of the input graph with the new edge attributes is returned. If the *inplace* parameter value is True, the graph is modified by setting the specified edge attributes.

set_node_attributes

```
set_node_attributes(values, name=None, immutable=None, *, inplace=False)
```

Sets node attributes from a given value or dictionary of values.

Parameters

values : scalar or dict-like or Pandas DataFrame
Specifies what the node attribute should be set to. When *values* is not a dictionary, it is treated as a single attribute value that is then applied to every node in *self*. This means that if you provide a mutable object, such as a list, updates to that object are reflected in the node attribute for every node. The attribute name is specified by the *name* parameter. When *values* is a dictionary, the keys are nodes and the values are either node attribute values or dictionaries of attribute name-value pairs.

name : str or None, optional
Specifies the name of the node attribute to set if the *values* parameter is set to a scalar. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not the graph is modified in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by setting the specified node attributes. If the *inplace* parameter value is False, a copy of the input graph with the new node attributes is returned.

shortest_path_distances

```
shortest_path_distances(source=None, sink=None, edge_weight=None,
minedgewt=None, maxedgewt=None)
```

Calculates the shortest path distances in a graph.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) and restricts enumeration to the shortest paths that contain the specified source node(s). The default is None.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) and restricts enumeration to the shortest paths that contain the specified sink node(s). The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

Returns

dict
Returns a dictionary where each key is a (source, sink) tuple and its value is the corresponding shortest source-sink path distance.

similarity_adamic_adar

```
similarity_adamic_adar(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Adamic-Adar node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_common_neighbors

```
similarity_common_neighbors(source, sink=None, top_k=None, bottom_k=None,
*, exclude_source=False, exclude_existing_neighbors=False)
```

Computes common neighbors node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_cosine

```
similarity_cosine(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes cosine node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_dice

```
similarity_dice(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Sørensen-Dice node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_graph

```
similarity_graph(measure, min_score=2.220446049250313e-16, max_score=inf,
*, exclude_source=False, exclude_existing_neighbors=False, data=True,
**kwargs)
```

Computes a node similarity graph.

Parameters

measure : str, one of {'adamic_adar', 'common_neighbors', 'cosine', 'dice', 'jaccard'}
Specifies the metric to be used to compute the node similarity.

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the edge attribute that corresponds to the measure name. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_adamic_adar

```
similarity_graph_adamic_adar(min_score=2.220446049250313e-16,
max_score=inf, *, exclude_source=False, exclude_existing_neighbors=False,
data=True)
```

Computes an Adamic-Adar node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'adamic_adar' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_common_neighbors

```
similarity_graph_common_neighbors(min_score=2.220446049250313e-16,
max_score=inf, *, exclude_source=False, exclude_existing_neighbors=False,
data=True)
```

Computes a common neighbors node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'common_neighbors' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_cosine

```
similarity_graph_cosine(edge_weight=None,
min_score=2.220446049250313e-16, max_score=inf, *, exclude_source=False,
exclude_existing_neighbors=False, data=True)
```

Computes a cosine node similarity graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'cosine' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_dice

```
similarity_graph_dice(min_score=2.220446049250313e-16, max_score=inf, *,
exclude_source=False, exclude_existing_neighbors=False, data=True)
```

Computes a Sørensen-Dice node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the *dice* edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_jaccard

```
similarity_graph_jaccard(min_score=2.220446049250313e-16, max_score=inf, *,
exclude_source=False, exclude_existing_neighbors=False, data=True)
```

Computes a Jaccard node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'jaccard' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_jaccard

```
similarity_jaccard(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Jaccard node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

singleton_node_count

```
singleton_node_count()
```

Computes a graph's singleton node count.

Returns

int
Returns the number of singleton nodes.

successors

```
successors(n)
```

Returns an iterator over all successor nodes of node n.

Parameters

n : node
Specifies a node in the graph.

Returns

iterator
Returns an iterator over all successor nodes of node n.

to_directed

```
to_directed()
```

Returns a directed copy of the input graph.

Returns

SAS graph object
Returns a directed copy of the input graph.

to_immutable

```
to_immutable()
```

Creates an immutable graph from the input graph.

Returns

SAS graph object
Returns an immutable SAS graph object.

to_mutable

```
to_mutable()
```

Creates a mutable graph from the input graph.

Returns

SAS graph object
Returns a mutable SAS graph object.

to_pandas_edges

```
to_pandas_edges(*, data=True, _for_conversion_to_immutable=False)
```

Returns the graph edges as a Pandas DataFrame.

Parameters

data : bool or list, optional
Specifies whether or not to retain attributes. The default is True. When the value is True (or when a list is given), the Pandas DataFrame includes all the edge attributes (or all the listed edge attributes).

Returns

dataframe
Returns a Pandas DataFrame that includes the graph edges.

to_pandas_nodes

```
to_pandas_nodes(*, data=True, _for_conversion_to_immutable=False)
```

Returns the graph nodes as a Pandas DataFrame.

Parameters

data : bool or list, optional
Specifies whether or not to retain attributes. The default is True. When the value is True (or when a list is given), the Pandas DataFrame includes all the node attributes (or all the listed node attributes).

Returns

dataframe
Returns a Pandas DataFrame that includes the graph nodes.

to_undirected

```
to_undirected()
```

Returns a directed copy of the input graph.

Returns

SAS graph object
Returns a directed copy of the input graph.

topological_ordering

```
topological_ordering()
```

Finds a topological ordering of the graph's nodes.

Yields

generator
Yields a generator of nodes in topological order. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

transitive_closure

```
transitive_closure()
```

Calculates the transitive closure of a graph.

Returns

SAS graph object
Returns the transitive closure of the input graph that is returned as a SAS MultiGraph or MultiDiGraph object. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

triangle_count

```
triangle_count()
```

Computes a graph's triangle count.

Returns

int
Returns the number of triangles.

tsp_tour

```
tsp_tour(edge_weight=None, maxtime=None, minobjective=None, *,
inplace=False, use_milp=True, data=True, **options)
```

Calculates a traveling salesman problem (TSP) tour of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted TSP is calculated.

maxtime : float or None, optional
Specifies the maximum amount of time for solving a TSP. The default is None.

minobjective : float, optional
Specifies a minimum value such that when a solution is found whose objective is less than or equal to this value, the algorithm stops.

inplace : bool, optional
Specifies whether to add the TSP order values as attributes to the input graph directly or to create a copy of the graph. The default is False.

use_milp : bool, optional
Specifies whether or not to use a mixed integer linear programming (MILP) solver to solve a TSP. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is False, the returned graph is a Cycle that contains the calculated TSP tour edges. When the *inplace* parameter value is True, the graph is modified in place and None is returned. The modified graph contains the edge attributes *tsp* and *tsp_order* and the node attribute *tsp_order*.

union

```
union(rhs, nodes_attrs_agg=None, edges_attrs_agg=None)
```

Returns the union of input graphs by finding the unions of their nodes and edges.

The node attributes and edge attributes are aggregated using the rules that are defined by the *nodes_attrs_agg* and *edges_attrs_agg* parameters.

Parameters

rhs : SAS graph object
Specifies the other graph, of the same type as *self*, with which to find the union.

nodes_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate node attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps node attributes to functions. The default is None.

edges_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate edge attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps edge attributes to functions. The default is None.

Returns

SAS graph object
Returns a graph that is constructed by the union of the two graphs.

vrp

```
vrp(capacity, depot, node_demand='demand', edge_cost='weight', minroutes=1,
maxroutes=None, maxtime=None, minobjective=None, *, use_milp=True,
data=True, **options)
```

Solves the vehicle routing problem.

Parameters

capacity : float
Specifies the capacity of each vehicle.

depot : node
Specifies the depot node for the vehicle routing problem.

node_demand : str or dict or iterable
Specifies the quantity of goods to deliver to or pick up from a customer. The default is 'demand'.

edge_cost : str or dict or iterable
Specifies the cost of traversing each edge. The default is 'weight'.

minroutes : int, optional
Specifies the minimum number of routes that are allowed to service demand.

maxroutes : int, optional
Specifies the maximum number of routes that are allowed to service demand.

maxtime : float or None, optional
Specifies the maximum amount of time for solving the vehicle routing problem. The default is None.

minobjective : float, optional
Specifies a minimum value such that when a solution is found whose objective is less than or equal to this value, the algorithm stops.

use_milp : bool, optional
Specifies whether or not to use a mixed integer linear programming (MILP) solver to solve the vehicle routing problem. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Cycle objects; each cycle represents one route. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

vrptw

```
vrptw(capacity, depot, node_demand='demand', edge_cost='weight',
travel_time='weight', timewindow_lower=None, timewindow_upper=None,
service_time=None, minroutes=1, maxroutes=None, maxtime=None, *,
data=True)
```

Solves the vehicle routing problem with time windows.

Parameters

capacity : float
Specifies the capacity of each vehicle.

depot : node
Specifies the depot node for the vehicle routing problem.

node_demand : str or dict or iterable
Specifies the quantity of goods to deliver to or pick up from a customer. The default is 'demand'.

edge_cost : str or dict or iterable
Specifies the cost of traversing each edge.

travel_time : str or dict or iterable
Specifies attributes or values that define the time that it takes to traverse each edge. The default is 'weight'.

timewindow_lower : str or dict or iterable or None, optional
Specifies attributes or values that define the lower time limits within which a delivery vehicle can arrive at the customer nodes. The default is None.

timewindow_upper : str or dict or iterable or None, optional
Specifies attributes or values that define the upper time limits within which a delivery vehicle can arrive at the customer nodes. The default is None.

service_time : str or dict or iterable or None, optional
Specifies attributes or values that define the service time of the nodes. The default is None.

minroutes : int, optional
Specifies the minimum number of routes that are allowed to service demand.

maxroutes : int, optional
Specifies the maximum number of routes that are allowed to service demand.

maxtime : float or None, optional
Specifies the maximum amount of time for solving the vehicle routing problem. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Cycle objects; each cycle represents one route. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

ImmutableGraph

Base class for undirected immutable graphs.

API: ImmutableGraph

class sasviya.network.classes.immutablegraph.ImmutableGraph(graph=None, **kwargs)

Self-loops and multiedges are not allowed.

Parameters

graph : SAS graph object
Specifies the graph to convert to an ImmutableGraph object.

Attributes

edges
Returns the edges of the graph.

nodes
Returns the nodes of the graph.

Methods

Method Name	Description
add_core_number_attr	Computes the core numbers and stores the results as node attributes on the graph.
add_topological_order_attr	Computes a topological ordering and stores the results as node attributes of the graph.

Method Name	Description
adjacency	Returns an iterator over (node, adjacency dict) tuples for all nodes.
allows_multiedges	Specifies whether or not the graph allows multiedges.
allows_selfloops	Specifies whether or not the graph allows self-loops.
approximate_diameter	Computes an approximation of the graph's diameter.
articulation_point_count	Computes a graph's articulation point count.
articulation_points	Finds the articulation points (cut points) of a graph.
assortativity_degree	Computes a graph's degree assortativity.
assortativity_nominal	Computes a graph's nominal assortativity.
assortativity_numeric	Computes a graph's numeric assortativity.
average_shortest_path	Computes a graph's average shortest path.
biconcomp_distribution_edges	Computes the distribution of the number of edges in each biconnected components by using *numpy.histogram*.
biconcomp_distribution_nodes	Computes the distribution of the number of nodes in each biconnected components by using *numpy.histogram*.
biconcomp_size_edges	Computes the number of edges within each biconnected component.
biconcomp_size_nodes	Computes the number of nodes within each biconnected component.
biconnected_component_count	Computes a graph's biconnected component count.
biconnected_components	Computes the graph's biconnected components for an undirected graph.
block_cut_tree	Computes the block-cut tree of a graph.
centrality_betweenness	Computes weighted or unweighted betweenness centrality and stores the results as attributes on the graph.
centrality_closeness	Computes weighted or unweighted closeness centrality and stores the results as attributes on the graph.

Method Name	Description
centrality_degree	Computes weighted or unweighted degree centrality and stores the results as attributes on the graph.
centrality_eigenvector	Computes weighted or unweighted eigenvector centrality and stores the results as attributes on the graph.
centrality_hub_authority	Computes weighted or unweighted hub and authority centrality and stores the results as attributes on the graph.
centrality_influence	Computes weighted or unweighted influence centrality and stores the results as attributes on the graph.
centrality_pagerank	Computes weighted or unweighted PageRank centrality and stores the results as attributes on the graph.
concomp_distribution_edges	Computes the distribution of the number of edges in each connected component by using *numpy.histogram*.
concomp_distribution_nodes	Computes the distribution of the number of nodes in each connected component by using *numpy.histogram*.
concomp_size_edges	Computes the number of edges within each connected component.
concomp_size_nodes	Computes the number of nodes within each connected component.
connected_component_count	Computes a graph's connected component count.
connected_components	Computes the connected components of a graph.
copy	Returns a copy of the input graph.
core_decomposition	Computes a core decomposition of the graph.
cycle_count	Counts the number of cycles in a graph.
degree	Returns the degree of nodes.
density	Computes the density of the graph.
diameter	Computes a graph's diameter.

Method Name	Description
drop_edge_attributes	Removes edge attributes if they exist.
drop_node_attributes	Removes node attributes if they exist.
enumerate_cycle_edges	Enumerates the cycles in a graph and returns the edges of the cycles.
enumerate_cycle_nodes	Enumerates the cycles in a graph and returns the nodes of the cycles.
enumerate_cycles	Enumerates the cycles in a graph and returns the cycle graph objects.
enumerate_path_edges	Enumerates the paths in a graph and returns the edges of the paths.
enumerate_path_nodes	Enumerates the paths in a graph and returns the nodes of the paths.
enumerate_paths	Enumerates the paths in a graph and returns the Path graph objects.
enumerate_sequence_shortest_paths	Enumerates the shortest paths in a graph that visit the indicated sequence nodes in order.
enumerate_shortest_path_edges	Enumerates the shortest paths in a graph and returns the edges of the paths.
enumerate_shortest_path_nodes	Enumerates the shortest paths in a graph and returns the nodes of the paths.
enumerate_shortest_paths	Enumerates the shortest paths in a graph and returns the Path graph objects.
get_meta_graph	Constructs a metagraph from the input graph.
get_nlargest_biconcomps	Returns a list of up to *n* largest biconnected components.
get_nlargest_concomps	Returns a list of up to *n* largest connected (with highest number of nodes) components of a graph.
has_edge_attribute	Checks the edges of the input graph for the specified edge attribute.
has_node_attribute	Checks the nodes of the input graph for the specified node attribute.
intersection	Returns the intersection of input graphs by finding the intersections of their nodes and edges.

Method Name	Description
is_biconnected	Specifies whether or not the input graph is biconnected.
is_bipartite	Determines whether the input graph is bipartite.
is_connected	Specifies whether or not the input graph is connected.
is_cycle	Determines whether the input graph is a simple cycle.
is_dag	Determines whether the input graph is a directed acyclic graph (DAG).
is_directed	Specifies whether or not the graph is directed.
is_hamiltonian	Determines whether the graph is Hamiltonian (has a Hamiltonian cycle) or not.
is_immutable	Specifies whether or not the graph is immutable.
is_multigraph	Specifies whether or not the graph allows multiedges.
is_path	Determines whether the input graph is a simple path.
is_strongly_connected	Specifies whether the input graph is strongly connected.
is_weakly_connected	Specifies whether the input graph is weakly connected.
label_propagation	Computes the communities of the graph by using the label propagation algorithm.
label_propagation_nodes	Computes community assignments for all nodes in the graph by using the label propagation algorithm.
leaf_node_count	Computes a graph's leaf node count.
local_transitivity	Computes the local transitivity (clustering coefficient) and stores the results as node attributes on the graph.
louvain	Computes the communities of the graph by using the Louvain algorithm.
louvain_nodes	Computes community assignments for all nodes in the graph by using the Louvain algorithm.

Method Name	Description
maxflow	Calculates the maximum flow from a source node to a sink node in a graph.
maximal_cliques	Enumerates all maximal cliques of the graph.
min_cut	Calculates a minimum cut of a graph.
min_s_t_cut	Calculates a minimum s-t cut of a graph.
mincostflow	Calculates the minimum-cost network flow for the graph.
minimum_spanning_tree	Calculates a minimum spanning tree (MST) or forest of a graph.
minimum_weight_full_matching	Returns a minimum-weight full matching of the bipartite graph *self*.
neighbors	Returns an iterator over all neighbors of node n.
node_clique_numbers	Computes node clique numbers and stores the results as attributes on the graph.
number_of_edges	Returns the number of edges in the graph.
number_of_nodes	Returns the number of nodes in the graph.
projected_graph	Computes a projected graph.
projected_graph_adamic_adar	Computes an Adamic-Adar projected graph.
projected_graph_common_neighbors	Computes a common neighbors projected graph.
projected_graph_cosine	Computes a cosine projected graph.
projected_graph_dice	Computes a Sørensen-Dice projected graph.
projected_graph_jaccard	Computes a Jaccard projected graph.
query	Queries for the subgraph isomorphisms of q within g.
reach_graph	Computes the reach graph for the source or set of sources.
reach_graph_per_source	Computes a separate reach graph for each given source node.
set_edge_attributes	Sets edge attributes by using a given value or dictionary of values.

Method Name	Description
set_node_attributes	Sets node attributes from a given value or dictionary of values.
shortest_path_distances	Calculates the shortest path distances in a graph.
similarity_adamic_adar	Computes Adamic-Adar node similarity.
similarity_common_neighbors	Computes common neighbors node similarity.
similarity_cosine	Computes cosine node similarity.
similarity_dice	Computes Sørensen-Dice node similarity.
similarity_graph	Computes a node similarity graph.
similarity_graph_adamic_adar	Computes an Adamic-Adar node similarity graph.
similarity_graph_common_neighbors	Computes a common neighbors node similarity graph.
similarity_graph_cosine	Computes a cosine node similarity graph.
similarity_graph_dice	Computes a Sørensen-Dice node similarity graph.
similarity_graph_jaccard	Computes a Jaccard node similarity graph.
similarity_jaccard	Computes Jaccard node similarity.
singleton_node_count	Computes a graph's singleton node count.
to_directed	Returns a directed copy of the input graph.
to_immutable	Creates an immutable graph from the input graph.
to_mutable	Creates a mutable graph from the input graph.
to_pandas_edges	Returns the graph edges as a Pandas DataFrame.
to_pandas_nodes	Returns the graph nodes as a Pandas DataFrame.
to_undirected	Returns a directed copy of the input graph.
topological_ordering	Finds a topological ordering of the graph's nodes.
transitive_closure	Calculates the transitive closure of a graph.
triangle_count	Computes a graph's triangle count.

Method Name	Description
tsp_tour	Calculates a traveling salesman problem (TSP) tour of a graph.
union	Returns the union of input graphs by finding the unions of their nodes and edges.
vrp	Solves the vehicle routing problem.
vrptw	Solves the vehicle routing problem with time windows.

add_core_number_attr

```
add_core_number_attr(maxtime=None, *, inplace=False, data=True)
```

Computes the core numbers and stores the results as node attributes on the graph.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time for core decomposition to spend. The default is None.

inplace : bool, optional
Specifies whether to add the core numbers as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph with core numbers that are stored as a node attribute named *core*. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

add_topological_order_attr

```
add_topological_order_attr(*, inplace=False, data=True)
```

Computes a topological ordering and stores the results as node attributes of the graph.

Parameters

inplace : bool, optional
Specifies whether or not to add the topological order values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a directed graph whose topological order is stored as a node attribute named *top_order*. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

adjacency

```
adjacency()
```

Returns an iterator over (node, adjacency dict) tuples for all nodes.

Returns

iterator
Returns an iterator of (node,adj dict) for all nodes.

Examples

```
G = sasnet.Graph([[1, 2, {'h': 2}],
                  [1, 3, {'w': 2, 'h': 0}],
                  [2, 3, {'w': 3, 'h': 5}]])
G_out = sasnet.ImmutableGraph(G)
```

```
print(list(G_out.adjacency()))
#[(1, {2: {'h': 2}, 3: {'w': 2, 'h': 0}}),
# (2, {1: {'h': 2}, 3: {'w': 3, 'h': 5}}),
# (3, {1: {'w': 2, 'h': 0}, 2: {'w': 3, 'h': 5}})].
```

allows_multiedges

```
allows_multiedges()
```

Specifies whether or not the graph allows multiedges.

Returns

bool
Indicates whether or not the graph allows multiedges.

allows_selfloops

```
allows_selfloops()
```

Specifies whether or not the graph allows self-loops.

Returns

bool
Indicates whether or not the graph allows self-loops.

approximate_diameter

```
approximate_diameter(edge_weight=None)
```

Computes an approximation of the graph's diameter.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted approximate diameter is computed.

Returns

float
Returns the approximate diameter.

articulation_point_count

```
articulation_point_count()
```

Computes a graph's articulation point count.

Returns

int
Returns the number of articulation points.

articulation_points

```
articulation_points()
```

Finds the articulation points (cut points) of a graph.

Returns

list
Returns a list of articulation points in the graph.

assortativity_degree

```
assortativity_degree(source_direction='out', target_direction='in',
edge_weight=None)
```

Computes a graph's degree assortativity.

Parameters

source_direction : {'in', 'out'}
Specifies the degree type (in-degree or out-degree) for the source node in a directed graph.

target_direction : {'in', 'out'}
Specifies the degree type (in-degree or out-degree) for the target node in a directed graph.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted assortativity degree is computed.

Returns

float
Returns the degree assortativity.

assortativity_nominal

```
assortativity_nominal(node_attr)
```

Computes a graph's nominal assortativity.

Parameters

node_attr : str
Specifies the name of the nominal node attribute to use.

Returns

float
Returns the nominal assortativity.

assortativity_numeric

```
assortativity_numeric(node_attr)
```

Computes a graph's numeric assortativity.

Parameters

node_attr : str
Specifies the name of the numeric node attribute to use.

Returns

float
Returns the numeric assortativity.

average_shortest_path

```
average_shortest_path(edge_weight=None)
```

Computes a graph's average shortest path.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted average shortest path is computed.

Returns

float
Returns the average shortest path.

biconcomp_distribution_edges

```
biconcomp_distribution_edges(bins=None)
```

Computes the distribution of the number of edges in each biconnected components by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is int (an integer), it specifies the number of equal-width bins. When *bins* is a list, it specifies bin ranges and should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is str (a string), it specifies the method to use for calculating the bin ranges. For example, 'stone','auto','doane','fd','rice','scott','sqrt','sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of biconnected components whose edge count is within the range of the bin.

biconcomp_distribution_nodes

```
biconcomp_distribution_nodes(bins=None)
```

Computes the distribution of the number of nodes in each biconnected components by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is int (an integer), it specifies the number of equal-width bins. When *bins* is a list, it specifies bin ranges and should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is str (a string), it specifies the method to use for calculating the bin ranges. For example, 'stone','auto','doane','fd','rice','scott','sqrt','sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of biconnected components whose node count is within the range of the bin.

biconcomp_size_edges

```
biconcomp_size_edges()
```

Computes the number of edges within each biconnected component.

Returns

dict
Returns a dictionary in which the keys are biconnected component IDs and the values are the number of edges within each component.

biconcomp_size_nodes

```
biconcomp_size_nodes()
```

Computes the number of nodes within each biconnected component.

Returns

dict
Returns a dictionary in which the keys are biconnected component IDs and the values are the number of nodes within each component.

biconnected_component_count

```
biconnected_component_count()
```

Computes a graph's biconnected component count.

Returns

int
Returns the number of biconnected components.

biconnected_components

```
biconnected_components(order_by=None, *, data=True, ascending=False)
```

Computes the graph's biconnected components for an undirected graph.

Parameters

order_by : 'node' or 'edge' or None, optional
Specifies the parameter that is used to sort the components by the number of nodes or edges within each component. The default is None, which means that the components are not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Returns the result as a generator of biconnected component subgraphs. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

block_cut_tree

```
block_cut_tree(*, return_nodes_map=False)
```

Computes the block-cut tree of a graph.

Parameters

return_nodes_map : bool, optional
Specifies whether or not to also compute the block-cut tree node map. The default is False.

Returns

SAS graph object
Returns a graph that represents the block-cut tree. When *return_nodes_map = True*, the output graph has the *nodes_map* attribute, which contains a dictionary that maps its nodes to a list of nodes in the original graph. Note: If a block is empty (that is, the corresponding biconnected component contains only articulation points), that block does not appear as a key in the *nodes_map* dictionary.

centrality_betweenness

```
centrality_betweenness(edge_weight=None, sample_percent=100, *,
inplace=False, data=True)
```

Computes weighted or unweighted betweenness centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that unweighted betweenness centrality is computed.

sample_percent : int, optional
Specifies the percentage of source nodes to sample for the approximate betweenness calculation. This parameter value should be in the range (0, 100]. The default is 100.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the node and edge attributes *between_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_closeness

```
centrality_closeness(edge_weight=None, nopath='diameter', *, inplace=False,
data=True)
```

Computes weighted or unweighted closeness centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted closeness centrality is computed.

nopath : {'diameter', 'harmonic', or 'zero'}, optional
Specifies a method to use for accounting for the shortest path distance between two nodes when a path does not exist. The valid values are as follows: - 'diameter' uses the graph diameter (plus one) as the shortest path distance between disconnected nodes. - 'harmonic' uses the harmonic formula for calculating closeness centrality. - 'zero' uses zero as the shortest path distance between disconnected nodes.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the closeness centrality node attributes *centr_close_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_degree

```
centrality_degree(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted degree centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted degree centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes

of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the degree centrality node attributes *centr_degree_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_eigenvector

```
centrality_eigenvector(edge_weight=None, maxiters=10000, *, inplace=False,
data=True)
```

Computes weighted or unweighted eigenvector centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted eigenvector centrality is computed.

maxiters : int, optional
Specifies the maximum number of iterations in order to limit the amount of computation time that is spent when convergence is slow. The default is 10,000.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the degree centrality node attributes *centr_eigen_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_hub_authority

```
centrality_hub_authority(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted hub and authority centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted hub and authority centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the hub and authority centrality node attributes *centr_hub_** and *centr_auth_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_influence

```
centrality_influence(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted influence centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted influence centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the first- and second-order influence centrality node attributes *centr_influence1_** and *centr_influence2_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_pagerank

```
centrality_pagerank(edge_weight=None, alpha=0.85, tolerance=1e-09, *,
inplace=False, data=True)
```

Computes weighted or unweighted PageRank centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted PageRank centrality is computed.

alpha : float, optional
Specifies the damping parameter for the PageRank centrality (or algorithm). The default is 0.85.

tolerance : float, optional
Specifies the tolerance parameter for the PageRank centrality (or algorithm). The default is 1E-9.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the PageRank centrality node attributes *centr_pagerank_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

concomp_distribution_edges

```
concomp_distribution_edges(bins=None, *, strongly=True)
```

Computes the distribution of the number of edges in each connected component by using *numpy.histogram*.

Parameters

bins : int or list or str, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is an integer (int), it represents the number of equal-width bins. When *bins* is a list, it specifies bin ranges and it should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is a string (str), it specifies the method to be used for calculating the bin ranges. For example, 'stone', 'auto', 'doane', 'fd', 'rice', 'scott', 'sqrt', 'sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of components whose edge count within the range of the bin.

concomp_distribution_nodes

```
concomp_distribution_nodes(bins=None, *, strongly=True)
```

Computes the distribution of the number of nodes in each connected component by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is an integer (int), it represents the number of equal-width bins. When *bins* is a list, it specifies bin ranges and it should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is a string (str), it specifies the method to use for calculating the bin ranges. For example, 'stone', 'auto', 'doane', 'fd', 'rice', 'scott', 'sqrt', 'sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of components whose node count is within the range of the bin.

concomp_size_edges

```
concomp_size_edges(*, strongly=True)
```

Computes the number of edges within each connected component.

Parameters

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary in which the keys are connected component IDs and the values are the number of edges within each component.

concomp_size_nodes

```
concomp_size_nodes(*, strongly=True)
```

Computes the number of nodes within each connected component.

Parameters

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary in which the keys are connected component IDs and the values are the number of nodes within each component.

connected_component_count

```
connected_component_count()
```

Computes a graph's connected component count.

Returns

int
Returns the number of connected components.

connected_components

```
connected_components(order_by=None, *, strongly=True, data=True,
ascending=False)
```

Computes the connected components of a graph.

Parameters

order_by : 'node' or 'edge' or None, optional
Specifies whether to sort the components by the number of nodes or edges within each component. The default is None, which means that the components are not sorted.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one connected component. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

copy

```
copy(immutable=None, *, data=True)
```

Returns a copy of the input graph.

Parameters

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, meaning that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns an independent copy of the input.

core_decomposition

```
core_decomposition(maxtime=None, *, data=True)
```

Computes a core decomposition of the graph.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time for core decomposition to spend. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one k-core in increasing core number. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

cycle_count

```
cycle_count(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Counts the number of cycles in a graph.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Returns

int
Returns a count of cycles in the graph.

degree

```
degree(n=None, edge_weight=None)
```

Returns the degree of nodes.

Parameters

n : single node or container of nodes or None, optional
Specifies which nodes to include. The default is None, which means that the degree of all nodes is returned.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None.

Returns

int or dictionary
Returns the degree of specified node(s).

density

```
density()
```

Computes the density of the graph.

Returns

float
Returns the density of the graph.

diameter

```
diameter(edge_weight=None)
```

Computes a graph's diameter.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted diameter is computed.

Returns

int or float
Returns the diameter.

drop_edge_attributes

```
drop_edge_attributes(attributes_subset=None, edges_subset=None,
immutable=None, *, inplace=False)
```

Removes edge attributes if they exist.

Parameters

attributes_subset : str or list, optional
Specifies the edge attribute or a list of the edge attributes to remove; for example, 'weight' or ['weight','color']. By default, all edge attributes are removed.

edges_subset : list or iterable of edges, optional
Specifies the edges to remove attributes from; for example, [(1,2),(2,3)]. By default the attributes for all edges are removed.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by removing the specified edge attributes. If the *inplace* parameter value is False, a copy of the input graph with the modified edge attributes is returned.

drop_node_attributes

```
drop_node_attributes(attributes_subset=None, nodes_subset=None,
immutable=None, *, inplace=False)
```

Removes node attributes if they exist.

Parameters

attributes_subset : str or list, optional
Specifies the node attribute or list of node attributes to remove; for example, ['weight','color']. By default, all node attributes are removed.

nodes_subset : list or iterable of nodes, optional
Specifies the nodes to drop attributes from; for example, [1,2,3] or (1,2,3) or [1] or (1,). By default, the attributes for all nodes are removed.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by removing the specified node attributes. If the *inplace* parameter value is False, a copy of the input graph with the modified node attributes is returned.

enumerate_cycle_edges

```
enumerate_cycle_edges(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Enumerates the cycles in a graph and returns the edges of the cycles.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Yields

generator
Yields a generator of cycles, each of which is given as a list of the cycles' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_cycle_nodes

```
enumerate_cycle_nodes(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Enumerates the cycles in a graph and returns the nodes of the cycles.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Yields

generator
Yields a generator of cycles, each of which is given as a list of the cycles' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_cycles

```
enumerate_cycles(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None, *,
data=True)
```

Enumerates the cycles in a graph and returns the cycle graph objects.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one cycle of the same type as the input graph. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_path_edges

```
enumerate_path_edges(source=None, sink=None, minlength=1,
maxlength=None, edge_weight=None, node_weight=None, minedgewt=None,
maxedgewt=None, minnodewt=None, maxnodewt=None, maxtime=None)
```

Enumerates the paths in a graph and returns the edges of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

Yields

generator
Yields a generator of Path objects, each of which is given as a list of the paths' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_path_nodes

```
enumerate_path_nodes(source=None, sink=None, minlength=1,
maxlength=None, edge_weight=None, node_weight=None, minedgewt=None,
maxedgewt=None, minnodewt=None, maxnodewt=None, maxtime=None)
```

Enumerates the paths in a graph and returns the nodes of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

Yields

generator
Yields a generator of Path objects, each of which is given as a list of the paths' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_paths

```
enumerate_paths(source=None, sink=None, minlength=1, maxlength=None,
edge_weight=None, node_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, *, data=True)
```

Enumerates the paths in a graph and returns the Path graph objects.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one Path object. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_sequence_shortest_paths

```
enumerate_sequence_shortest_paths(sequence, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None, *, data=True)
```

Enumerates the shortest paths in a graph that visit the indicated sequence nodes in order.

Parameters

sequence : iterable of nodes
Specifies the order in which to visit the required nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Path objects; each object represents a subpath between a sequence of nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_path_edges

```
enumerate_shortest_path_edges(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None)
```

Enumerates the shortest paths in a graph and returns the edges of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

Yields

generator
Yields a generator of paths; each path is given as a list of the paths' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_path_nodes

```
enumerate_shortest_path_nodes(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None)
```

Enumerates the shortest paths in a graph and returns the nodes of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

Yields

generator
Yields a generator of paths; each path is given as a list of the paths' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_paths

```
enumerate_shortest_paths(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None, *, data=True)
```

Enumerates the shortest paths in a graph and returns the Path graph objects.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Path objects; each object represents one shortest path. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

get_meta_graph

```
get_meta_graph(node_type, edge_type=None, immutable=None)
```

Constructs a metagraph from the input graph.

Parameters

node_type : str
Specifies the node attribute to consider in constructing the metagraph.

edge_type : str, optional
Specifies the edge attribute to consider in constructing the metagraph. By default, all connections between nodes are considered the same.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

Returns

SAS graph object
Returns the metagraph, which has an edge attribute count and determines the number of unique connections between pairs of nodes.

get_nlargest_biconcomps

```
get_nlargest_biconcomps(n, order_by='node', *, data=True, ascending=False)
```

Returns a list of up to *n* largest biconnected components.

Parameters

n : int
Specifies the number of graphs to return.

order_by : 'node' or 'edge' or None
Specifies whether to sort the biconnected components by the number of nodes or edges within those components. The default is None, which means that the biconnected components are not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. When the value is True, the smallest biconnected components are returned instead. The default is False.

Returns

list
Returns a list of n largest or smallest biconnected components of a graph.

get_nlargest_concomps

```
get_nlargest_concomps(n, order_by='node', *, strongly=True, data=True,
ascending=False)
```

Returns a list of up to *n* largest connected (with highest number of nodes) components of a graph.

Parameters

n : int
Specifies the number of graphs to return.

order_by : 'node' or 'edge'
Specifies whether to sort the component by number of nodes or edges within that component. The default is None, which means that the result is not sorted.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Returns

list
Returns a list of the n largest connected components of a graph.

has_edge_attribute

```
has_edge_attribute(name, *, all_edges=False)
```

Checks the edges of the input graph for the specified edge attribute.

Parameters

name : str
Specifies the name of the edge attribute.

all_edges : bool, optional
Specifies whether or not to check for the existence of the edge attribute in all or any of the edges. The default is False.

Returns

bool
If the all_edges parameter value is True, the value True is returned if that edge attribute exists in all edges; otherwise the value False is returned. If the all_edges parameter value is False, the value True is returned if at least one edge in the graph has that specified edge attribute.

has_node_attribute

```
has_node_attribute(name, *, all_nodes=False)
```

Checks the nodes of the input graph for the specified node attribute.

Parameters

name : str
Specifies the name of the node attribute.

all_nodes : bool, optional
Specifies whether or not to check for the existence of the node attribute in all or any of the nodes. The default is False.

Returns

bool
If the all_nodes parameter value is True, the value True is returned if that node attribute exists in all node; otherwise the value False is returned. If the

all_nodes parameter value is False, the value True is returned if at least one node in the graph has that specified node attribute.

intersection

```
intersection(rhs, nodes_attrs_agg=None, edges_attrs_agg=None)
```

Returns the intersection of input graphs by finding the intersections of their nodes and edges.

The node attributes and edge attributes are aggregated using the rules that are defined by the *nodes_attrs_agg* and *edges_attrs_agg* parameters.

Parameters

rhs : SAS graph object
Specifies the other graph, of the same type as *self*, with which to intersect.

nodes_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate node attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps node attributes to functions. The default is None.

edges_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate edge attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps edge attributes to functions. The default is None.

Returns

SAS graph object
Returns a graph that is constructed by the intersection of the two graphs.

is_biconnected

```
is_biconnected()
```

Specifies whether or not the input graph is biconnected.

Returns

bool
Indicates whether the input graph is biconnected (True) or not (False).

is_bipartite

```
is_bipartite()
```

Determines whether the input graph is bipartite.

Returns

bool
Indicates whether the input graph is bipartite.

is_connected

```
is_connected()
```

Specifies whether or not the input graph is connected.

Returns

bool
Indicates whether the input graph is connected (True) or not (False).

is_cycle

```
is_cycle()
```

Determines whether the input graph is a simple cycle.

Returns

bool
Indicates whether the input graph is a simple cycle (True) or not (False).

is_dag

```
is_dag()
```

Determines whether the input graph is a directed acyclic graph (DAG).

Returns

bool
Indicates whether the input graph is a directed acyclic graph (DAG).

is_directed

```
is_directed()
```

Specifies whether or not the graph is directed.

Returns

bool
Indicates whether or not the graph is directed.

is_hamiltonian

```
is_hamiltonian(maxtime=None)
```

Determines whether the graph is Hamiltonian (has a Hamiltonian cycle) or not.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time to determine whether or not the graph is Hamiltonian. The default is None.

Returns

bool or None
Indicates whether or not the input graph is determined to be Hamiltonian. Returns None if no determination is made.

is_immutable

```
is_immutable()
```

Specifies whether or not the graph is immutable.

Returns

bool
Indicates whether or not the graph is immutable.

is_multigraph

```
is_multigraph()
```

Specifies whether or not the graph allows multiedges.

Returns

bool
Indicates whether or not the graph allows multiedges.

is_path

```
is_path()
```

Determines whether the input graph is a simple path.

Returns

bool
Indicates whether the input graph is a simple path (True) or not (False).

is_strongly_connected

```
is_strongly_connected()
```

Specifies whether the input graph is strongly connected.

Returns

bool
Indicates whether the input graph is strongly connected (True) or not (False).

is_weakly_connected

```
is_weakly_connected()
```

Specifies whether the input graph is weakly connected.

Returns

bool
Indicates whether the input graph is weakly connected (True) or not (False).

label_propagation

```
label_propagation(edge_weight=None, maxiters=100, tolerance=0.05, *,
data=True)
```

Computes the communities of the graph by using the label propagation algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of label changes for all nodes in the graph is less than the value. The default is 0.05.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one community. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

label_propagation_nodes

```
label_propagation_nodes(edge_weight=None, maxiters=100, tolerance=0.05, *,
data=True, inplace=False)
```

Computes community assignments for all nodes in the graph by using the label propagation algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of label changes for all nodes in the graph is less than the value. The default is 0.05.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

inplace : bool, optional
Specifies whether to add the community values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains the node attribute *community*, which denotes the community assignment for each node. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

leaf_node_count

```
leaf_node_count()
```

Computes a graph's leaf node count.

Returns

int
Returns the number of leaf nodes.

local_transitivity

```
local_transitivity(*, inplace=False)
```

Computes the local transitivity (clustering coefficient) and stores the results as node attributes on the graph.

Parameters

inplace : bool, optional
Specifies whether to add the transitivity values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains node attribute 'transitivity'. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

louvain

```
louvain(*, edge_weight=None, resolution=1, tolerance=0.001, maxiters=100,
data=True)
```

Computes the communities of the graph by using the Louvain algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

resolution : float, optional
Specifies the factor in the modularity equation that affects the expected number of links between two nodes. A higher value produces more communities. The default is 1.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of modularity gain between two consecutive iterations is less than the value. The default is 0.001.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one community. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

louvain_nodes

```
louvain_nodes(*, edge_weight=None, resolution=1, tolerance=0.001,
maxiters=100, data=True, inplace=False)
```

Computes community assignments for all nodes in the graph by using the Louvain algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

resolution : float, optional
Specifies the factor in the modularity equation that affects the expected number of links between two nodes. A higher value produces more communities. The default is 1.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of modularity gain between two consecutive iterations is less than the value. The default is 0.001.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

inplace : bool, optional
Specifies whether to add the community values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains the node attribute *community*, which denotes the community assignment for each node. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

maxflow

```
maxflow(source, sink, capacity_attr='upper', *, inplace=False, data=True)
```

Calculates the maximum flow from a source node to a sink node in a graph.

Parameters

source : number or str
Specifies the source node for the maximum flow.

sink : number or str
Specifies the sink node for the maximum flow.

capacity_attr : str, optional
Specifies the name of the edge attribute that contains the capacity of the edge. The default is 'upper'.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute *flow*, which contains the flow values. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, the returned graph contains only the flow edges.

maximal_cliques

```
maximal_cliques(maxcliques='ALL', node_weight=None, edge_weight=None,
minsize=1, maxsize=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, order_by=None, *,
data=True, ascending=False)
```

Enumerates all maximal cliques of the graph.

Parameters

maxcliques : int or 'ALL', optional
Specifies the maximum number of enumerated cliques to return. The default is 'ALL'.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

minsize : int, optional
Specifies the minimum number of nodes in a clique. The default is 1.

maxsize : int or None, optional
Specifies the maximum number of nodes in a clique. The default is None.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a clique. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a clique. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a clique. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a clique. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds for enumeration to spend. The default is None.

order_by : 'node' or None, optional
Specifies whether to sort the cliques by the number of nodes within each clique. The default is None, which means that the result is not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one maximal clique. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

min_cut

```
min_cut(edge_weight=None, *, inplace=False, data=True)
```

Calculates a minimum cut of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted minimum cut is calculated.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes

of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute 'mincut', which indicates the edges to include in the minimum cut set, and the node attribute 'mincut_partition', which indicates the node partition of the calculated minimum cut. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, a modified copy of the graph is returned.

min_s_t_cut

```
min_s_t_cut(source, sink, edge_weight=None, *, inplace=False, data=True)
```

Calculates a minimum s-t cut of a graph.

Parameters

source : number or str
Specifies the source node (s) for the minimum s-t cut.

sink : number or str
Specifies the sink node (t) for the minimum s-t cut.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted minimum s-t cut is calculated.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute *s_t_cut*, which indicates the edges to include in the minimum s-t cut set; and the node attribute *s_t_cut_partition*, which indicates the node partition of the calculated minimum s-t cut. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is

modified in place and None is returned. When the *inplace* parameter value is False, a modified copy of the graph is returned.

mincostflow

```
mincostflow(demand_lower_attr='lower', demand_upper_attr='upper',
capacity_lower_attr='lower', capacity_upper_attr='upper', cost_attr='weight',
maxtime=None, *, inplace=False, data=True)
```

Calculates the minimum-cost network flow for the graph.

Parameters

demand_lower_attr : str, optional
Specifies the name of the node attribute that contains the lower bound of the node supply. The default is 'lower'.

demand_upper_attr : str, optional
Specifies the name of the node attribute that contains the upper bound of the node supply. The default is 'upper'.

capacity_lower_attr : str, optional
Specifies the name of the edge attribute that contains the capacity lower bound of the edge. The default is 'lower'.

capacity_upper_attr : str, optional
Specifies the name of the edge attribute that contains the capacity upper bound of the edge. The default is 'upper'.

cost_attr : str, optional
Specifies the name of the edge attribute that contains the edge cost. The default is 'weight'.

maxtime : float or None, optional
Specifies the maximum amount of time to allow for computing the minimum-cost network flow. The default is None.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attributes *flow* and *reduced_cost*, which contain the flow and reduced cost values, respectively; and the node attribute *dual*, which contains the optimal dual value. This graph has the

solution_summary attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned.

minimum_spanning_tree

```
minimum_spanning_tree(edge_weight=None, source=None, *, inplace=False,
data=True)
```

Calculates a minimum spanning tree (MST) or forest of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

source : number or str or None
Specifies the source node for a directed MST. The default is None.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute 'mst', which indicates the edges to include in the spanning tree or forest. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, the returned graph contains only the spanning tree or forest edges.

minimum_weight_full_matching

```
minimum_weight_full_matching(edge_weight=None, *, data=True)
```

Returns a minimum-weight full matching of the bipartite graph *self*.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns a directed graph whose edges identify the assignment mapping in the minimum-weight full matching solution. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

neighbors

```
neighbors(n)
```

Returns an iterator over all neighbors of node n.

Parameters

n : node
Specifies a node in the graph.

Returns

iterator
Returns an iterator over all neighbors of node n.

node_clique_numbers

```
node_clique_numbers(*, inplace=False, data=True)
```

Computes node clique numbers and stores the results as attributes on the graph.

Parameters

inplace : bool, optional
Specifies whether to add the clique number values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the value of the *inplace* parameter is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph or modifies the existing graph to contain the node attribute *clique_number*, which includes the clique number of each node. The graph g has the attribute g.solution_summary, which contains information about the results of the algorithm.

number_of_edges

```
number_of_edges(u=None, v=None)
```

Returns the number of edges in the graph.

When you specify *u* and *v*, the number of edges between *u* and *v* is returned. When you specify *u* but *v* is None, the number of edges that connect to *u* is returned. When you specify *v* but *u* is None, the number of edges that connect to *v* is returned.

Parameters

u : numeric or str or None, optional
Specifies the *from* node. The default is None.

v : numeric or str or None, optional
Specifies the *to* node. The default is None.

Returns

int
Returns the number of edges in the graph. When you specify *u* and *v*, the number of edges between *u* and *v* is returned. When you specify *u* but *v* is None, the number of edges that connect to *u* is returned. When you specify *v* but *u* is None, the number of edges that connect to *v* is returned.

number_of_nodes

```
number_of_nodes()
```

Returns the number of nodes in the graph.

Returns

int
Returns the number of nodes in the graph.

projected_graph

```
projected_graph(nodes, neighbors=None, directed_method=None,
_measure=None, *, multigraph=False, data=True)
```

Computes a projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in self other than *nodes*.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used. When the *directed_method* parameter is specified for an undirected graph, a ValueError is raised.

multigraph : bool, optional
Specifies whether or not to return a multigraph. When the value is True, the algorithm returns a multigraph in which the multiple edges represent multiple shared neighbors. The edge attribute 'neighbor' stores the label of the shared neighbor. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

projected_graph_adamic_adar

```
projected_graph_adamic_adar(nodes, neighbors=None, directed_method=None,
*, data=True)
```

Computes an Adamic-Adar projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than *nodes*.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'adamic_adar' edge attribute.

projected_graph_common_neighbors

```
projected_graph_common_neighbors(nodes, neighbors=None,
directed_method=None, *, data=True)
```

Computes a common neighbors projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'common_neighbors' edge attribute.

projected_graph_cosine

```
projected_graph_cosine(nodes, edge_weight=None, neighbors=None,
directed_method=None, *, data=True)
```

Computes a cosine projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'cosine' edge attribute.

projected_graph_dice

```
projected_graph_dice(nodes, neighbors=None, directed_method=None, *,
data=True)
```

Computes a Sørensen-Dice projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the subset of nodes that are specified by the *neighbors* parameter. The strength of association in the calculated projection is represented by the 'dice' edge attribute.

projected_graph_jaccard

```
projected_graph_jaccard(nodes, neighbors=None, directed_method=None, *,
data=True)
```

Computes a Jaccard projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'jaccard' edge attribute.

query

```
query(q, expr=None, _node_attr_to_ignore=None, _edge_attr_to_ignore=None, *,
induced=False, break_symmetry=False, _generate_output=True,
_orbits_only=False, **kwargs)
```

Queries for the subgraph isomorphisms of q within g.

Parameters

q : graph
Specifies the query graph.

expr : _Expression or None, optional
Specifies additional logic to filter the query results. The default is None.

induced : bool, optional
Specifies whether or not to return only node-induced matches. The default is False.

break_symmetry : bool, optional
Specifies whether or not to break symmetry and eliminate automorphic relationships between matches. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one isomorphic mapping from *q* to *self*. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

reach_graph

```
reach_graph(source, max_reach=1, *, data=True)
```

Computes the reach graph for the source or set of sources.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to compute the reach network.

max_reach : int, optional
Specifies the number of steps (or hops) to traverse from the source nodes. The default is 1.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns a graph that represents the reach network. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

reach_graph_per_source

```
reach_graph_per_source(source, max_reach=1, *, data=True)
```

Computes a separate reach graph for each given source node.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to compute the reach network.

max_reach : int, optional
Specifies the number of steps (or hops) to traverse from the source nodes. The default is 1.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one reach network. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

set_edge_attributes

```
set_edge_attributes(values, name=None, immutable=None, *, inplace=False)
```

Sets edge attributes by using a given value or dictionary of values.

Parameters

values : scalar or dict-like or Pandas DataFrame
Specifies what the edge attribute should be set to. When this parameter is set to a Pandas DataFrame, it is expected to have from, to, and edge_key (only for multigraphs) columns, and the other columns are treated as edge attributes to add. When *values* is a dictionary, the keys are edges and the values are either edge attribute values or dictionaries of attribute name-value pairs. For multigraphs, the edge tuples must be of the form (u, v, key). For simple graphs, the keys must be tuples of the form (u, v). When this parameter is not set to a dictionary, it is treated as a single attribute value that is then applied to every edge in *self*. The attribute name is specified by the *name* parameter.

name : str or None, optional
Specifies the name of the edge attribute to set if the *values* parameter is set to a scalar. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is False, a copy of the input graph with the new edge attributes is returned. If the *inplace* parameter value is True, the graph is modified by setting the specified edge attributes.

set_node_attributes

```
set_node_attributes(values, name=None, immutable=None, *, inplace=False)
```

Sets node attributes from a given value or dictionary of values.

Parameters

values : scalar or dict-like or Pandas DataFrame
Specifies what the node attribute should be set to. When *values* is not a dictionary, it is treated as a single attribute value that is then applied to every node in *self*. This means that if you provide a mutable object, such as a list, updates to that object are reflected in the node attribute for every node. The attribute name is specified by the *name* parameter. When *values* is a dictionary, the keys are nodes and the values are either node attribute values or dictionaries of attribute name-value pairs.

name : str or None, optional
Specifies the name of the node attribute to set if the *values* parameter is set to a scalar. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not the graph is modified in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by setting the specified node attributes. If the *inplace* parameter value is False, a copy of the input graph with the new node attributes is returned.

shortest_path_distances

```
shortest_path_distances(source=None, sink=None, edge_weight=None,
minedgewt=None, maxedgewt=None)
```

Calculates the shortest path distances in a graph.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) and restricts enumeration to the shortest paths that contain the specified source node(s). The default is None.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) and restricts enumeration to the shortest paths that contain the specified sink node(s). The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

Returns

dict
Returns a dictionary where each key is a (source, sink) tuple and its value is the corresponding shortest source-sink path distance.

similarity_adamic_adar

```
similarity_adamic_adar(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Adamic-Adar node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_common_neighbors

```
similarity_common_neighbors(source, sink=None, top_k=None, bottom_k=None,
*, exclude_source=False, exclude_existing_neighbors=False)
```

Computes common neighbors node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_cosine

```
similarity_cosine(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes cosine node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_dice

```
similarity_dice(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Sørensen-Dice node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_graph

```
similarity_graph(measure, min_score=2.220446049250313e-16, max_score=inf,
*, exclude_source=False, exclude_existing_neighbors=False, data=True,
**kwargs)
```

Computes a node similarity graph.

Parameters

measure : str, one of {'adamic_adar', 'common_neighbors', 'cosine', 'dice', 'jaccard'}
Specifies the metric to be used to compute the node similarity.

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the edge attribute that corresponds to the measure name. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_adamic_adar

```
similarity_graph_adamic_adar(min_score=2.220446049250313e-16,
max_score=inf, *, exclude_source=False, exclude_existing_neighbors=False,
data=True)
```

Computes an Adamic-Adar node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'adamic_adar' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_common_neighbors

```
similarity_graph_common_neighbors(min_score=2.220446049250313e-16,
max_score=inf, *, exclude_source=False, exclude_existing_neighbors=False,
data=True)
```

Computes a common neighbors node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'common_neighbors' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_cosine

```
similarity_graph_cosine(edge_weight=None,
min_score=2.220446049250313e-16, max_score=inf, *, exclude_source=False,
exclude_existing_neighbors=False, data=True)
```

Computes a cosine node similarity graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'cosine' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_dice

```
similarity_graph_dice(min_score=2.220446049250313e-16, max_score=inf, *,
exclude_source=False, exclude_existing_neighbors=False, data=True)
```

Computes a Sørensen-Dice node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the *dice* edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_jaccard

```
similarity_graph_jaccard(min_score=2.220446049250313e-16, max_score=inf, *,
exclude_source=False, exclude_existing_neighbors=False, data=True)
```

Computes a Jaccard node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'jaccard' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_jaccard

```
similarity_jaccard(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Jaccard node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

singleton_node_count

```
singleton_node_count()
```

Computes a graph's singleton node count.

Returns

int
Returns the number of singleton nodes.

to_directed

```
to_directed()
```

Returns a directed copy of the input graph.

Returns

SAS graph object
Returns a directed copy of the input graph.

to_immutable

```
to_immutable()
```

Creates an immutable graph from the input graph.

Returns

SAS graph object
Returns an immutable SAS graph object.

to_mutable

```
to_mutable()
```

Creates a mutable graph from the input graph.

Returns

SAS graph object
Returns a mutable SAS graph object.

to_pandas_edges

```
to_pandas_edges(*, data=True, _for_conversion_to_immutable=False)
```

Returns the graph edges as a Pandas DataFrame.

Parameters

data : bool or list, optional
Specifies whether or not to retain attributes. The default is True. When the value is True (or when a list is given), the Pandas DataFrame includes all the edge attributes (or all the listed edge attributes).

Returns

dataframe
Returns a Pandas DataFrame that includes the graph edges.

to_pandas_nodes

```
to_pandas_nodes(*, data=True, _for_conversion_to_immutable=False)
```

Returns the graph nodes as a Pandas DataFrame.

Parameters

data : bool or list, optional
Specifies whether or not to retain attributes. The default is True. When the value is True (or when a list is given), the Pandas DataFrame includes all the node attributes (or all the listed node attributes).

Returns

dataframe
Returns a Pandas DataFrame that includes the graph nodes.

to_undirected

```
to_undirected()
```

Returns a directed copy of the input graph.

Returns

SAS graph object
Returns a directed copy of the input graph.

topological_ordering

```
topological_ordering()
```

Finds a topological ordering of the graph's nodes.

Yields

generator
Yields a generator of nodes in topological order. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

transitive_closure

```
transitive_closure()
```

Calculates the transitive closure of a graph.

Returns

SAS graph object
Returns the transitive closure of the input graph that is returned as a SAS MultiGraph or MultiDiGraph object. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

triangle_count

```
triangle_count()
```

Computes a graph's triangle count.

Returns

int
Returns the number of triangles.

tsp_tour

```
tsp_tour(edge_weight=None, maxtime=None, minobjective=None, *,
inplace=False, use_milp=True, data=True, **options)
```

Calculates a traveling salesman problem (TSP) tour of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted TSP is calculated.

maxtime : float or None, optional
Specifies the maximum amount of time for solving a TSP. The default is None.

minobjective : float, optional
Specifies a minimum value such that when a solution is found whose objective is less than or equal to this value, the algorithm stops.

inplace : bool, optional
Specifies whether to add the TSP order values as attributes to the input graph directly or to create a copy of the graph. The default is False.

use_milp : bool, optional
Specifies whether or not to use a mixed integer linear programming (MILP) solver to solve a TSP. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is False, the returned graph is a Cycle that contains the calculated TSP tour edges. When the *inplace* parameter value is True, the graph is modified in place and None is returned. The modified graph contains the edge attributes *tsp* and *tsp_order* and the node attribute *tsp_order*.

union

```
union(rhs, nodes_attrs_agg=None, edges_attrs_agg=None)
```

Returns the union of input graphs by finding the unions of their nodes and edges.

The node attributes and edge attributes are aggregated using the rules that are defined by the *nodes_attrs_agg* and *edges_attrs_agg* parameters.

Parameters

rhs : SAS graph object
Specifies the other graph, of the same type as *self*, with which to find the union.

nodes_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate node attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps node attributes to functions. The default is None.

edges_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate edge attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps edge attributes to functions. The default is None.

Returns

SAS graph object
Returns a graph that is constructed by the union of the two graphs.

vrp

```
vrp(capacity, depot, node_demand='demand', edge_cost='weight', minroutes=1,
maxroutes=None, maxtime=None, minobjective=None, *, use_milp=True,
data=True, **options)
```

Solves the vehicle routing problem.

Parameters

capacity : float
Specifies the capacity of each vehicle.

depot : node
Specifies the depot node for the vehicle routing problem.

node_demand : str or dict or iterable
Specifies the quantity of goods to deliver to or pick up from a customer. The default is 'demand'.

edge_cost : str or dict or iterable
Specifies the cost of traversing each edge. The default is 'weight'.

minroutes : int, optional
Specifies the minimum number of routes that are allowed to service demand.

maxroutes : int, optional
Specifies the maximum number of routes that are allowed to service demand.

maxtime : float or None, optional
Specifies the maximum amount of time for solving the vehicle routing problem. The default is None.

minobjective : float, optional
Specifies a minimum value such that when a solution is found whose objective is less than or equal to this value, the algorithm stops.

use_milp : bool, optional
Specifies whether or not to use a mixed integer linear programming (MILP) solver to solve the vehicle routing problem. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Cycle objects; each cycle represents one route. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

vrptw

```
vrptw(capacity, depot, node_demand='demand', edge_cost='weight',
travel_time='weight', timewindow_lower=None, timewindow_upper=None,
service_time=None, minroutes=1, maxroutes=None, maxtime=None, *,
data=True)
```

Solves the vehicle routing problem with time windows.

Parameters

capacity : float
Specifies the capacity of each vehicle.

depot : node
Specifies the depot node for the vehicle routing problem.

node_demand : str or dict or iterable
Specifies the quantity of goods to deliver to or pick up from a customer. The default is 'demand'.

edge_cost : str or dict or iterable
Specifies the cost of traversing each edge.

travel_time : str or dict or iterable
Specifies attributes or values that define the time that it takes to traverse each edge. The default is 'weight'.

timewindow_lower : str or dict or iterable or None, optional
Specifies attributes or values that define the lower time limits within which a delivery vehicle can arrive at the customer nodes. The default is None.

timewindow_upper : str or dict or iterable or None, optional
Specifies attributes or values that define the upper time limits within which a delivery vehicle can arrive at the customer nodes. The default is None.

service_time : str or dict or iterable or None, optional
Specifies attributes or values that define the service time of the nodes. The default is None.

minroutes : int, optional
Specifies the minimum number of routes that are allowed to service demand.

maxroutes : int, optional
Specifies the maximum number of routes that are allowed to service demand.

maxtime : float or None, optional
Specifies the maximum amount of time for solving the vehicle routing problem. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Cycle objects; each cycle represents one route. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

ImmutableMultiDiGraph

Base class for directed immutable multigraphs.

API: ImmutableMultiDiGraph

class
sasviya.network.classes.immutablemultidigraph.ImmutableMultiDiGraph(graph=None, **kwargs)

Self-loops and multiedges are allowed.

Parameters

graph : sasnet.MultiDiGraph object
Specifies the graph to convert to an ImmutableMultiDiGraph object.

Attributes

edges
Returns the edges of the graph.

nodes
Returns the nodes of the graph.

Methods

Method Name	Description
add_core_number_attr	Computes the core numbers and stores the results as node attributes on the graph.
add_topological_order_attr	Computes a topological ordering and stores the results as node attributes of the graph.
adjacency	Returns an iterator over (node, adjacency dict) tuples for all nodes.
allows_multiedges	Specifies whether or not the graph allows multiedges.
allows_selfloops	Specifies whether or not the graph allows self-loops.
approximate_diameter	Computes an approximation of the graph's diameter.
articulation_point_count	Computes a graph's articulation point count.
articulation_points	Finds the articulation points (cut points) of a graph.
assortativity_degree	Computes a graph's degree assortativity.
assortativity_nominal	Computes a graph's nominal assortativity.
assortativity_numeric	Computes a graph's numeric assortativity.
average_shortest_path	Computes a graph's average shortest path.
biconcomp_distribution_edges	Computes the distribution of the number of edges in each biconnected components by using *numpy.histogram*.
biconcomp_distribution_nodes	Computes the distribution of the number of nodes in each biconnected components by using *numpy.histogram*.

Method Name	Description
biconcomp_size_edges	Computes the number of edges within each biconnected component.
biconcomp_size_nodes	Computes the number of nodes within each biconnected component.
biconnected_component_count	Computes a graph's biconnected component count.
biconnected_components	Computes the graph's biconnected components for an undirected graph.
block_cut_tree	Computes the block-cut tree of a graph.
centrality_betweenness	Computes weighted or unweighted betweenness centrality and stores the results as attributes on the graph.
centrality_closeness	Computes weighted or unweighted closeness centrality and stores the results as attributes on the graph.
centrality_degree	Computes weighted or unweighted degree centrality and stores the results as attributes on the graph.
centrality_eigenvector	Computes weighted or unweighted eigenvector centrality and stores the results as attributes on the graph.
centrality_hub_authority	Computes weighted or unweighted hub and authority centrality and stores the results as attributes on the graph.
centrality_influence	Computes weighted or unweighted influence centrality and stores the results as attributes on the graph.
centrality_pagerank	Computes weighted or unweighted PageRank centrality and stores the results as attributes on the graph.
concomp_distribution_edges	Computes the distribution of the number of edges in each connected component by using *numpy.histogram*.
concomp_distribution_nodes	Computes the distribution of the number of nodes in each connected component by using *numpy.histogram*.

Method Name	Description
concomp_size_edges	Computes the number of edges within each connected component.
concomp_size_nodes	Computes the number of nodes within each connected component.
connected_component_count	Computes a graph's connected component count.
connected_components	Computes the connected components of a graph.
copy	Returns a copy of the input graph.
core_decomposition	Computes a core decomposition of the graph.
cycle_count	Counts the number of cycles in a graph.
degree	Returns the degree of nodes.
density	Computes the density of the graph.
diameter	Computes a graph's diameter.
drop_edge_attributes	Removes edge attributes if they exist.
drop_node_attributes	Removes node attributes if they exist.
enumerate_cycle_edges	Enumerates the cycles in a graph and returns the edges of the cycles.
enumerate_cycle_nodes	Enumerates the cycles in a graph and returns the nodes of the cycles.
enumerate_cycles	Enumerates the cycles in a graph and returns the cycle graph objects.
enumerate_path_edges	Enumerates the paths in a graph and returns the edges of the paths.
enumerate_path_nodes	Enumerates the paths in a graph and returns the nodes of the paths.
enumerate_paths	Enumerates the paths in a graph and returns the Path graph objects.
enumerate_sequence_shortest_paths	Enumerates the shortest paths in a graph that visit the indicated sequence nodes in order.
enumerate_shortest_path_edges	Enumerates the shortest paths in a graph and returns the edges of the paths.

Method Name	Description
enumerate_shortest_path_nodes	Enumerates the shortest paths in a graph and returns the nodes of the paths.
enumerate_shortest_paths	Enumerates the shortest paths in a graph and returns the Path graph objects.
get_meta_graph	Constructs a metagraph from the input graph.
get_nlargest_biconcomps	Returns a list of up to *n* largest biconnected components.
get_nlargest_concomps	Returns a list of up to *n* largest connected (with highest number of nodes) components of a graph.
has_edge_attribute	Checks the edges of the input graph for the specified edge attribute.
has_node_attribute	Checks the nodes of the input graph for the specified node attribute.
in_degree	Returns the in-degree of nodes.
intersection	Returns the intersection of input graphs by finding the intersections of their nodes and edges.
is_biconnected	Specifies whether or not the input graph is biconnected.
is_bipartite	Determines whether the input graph is bipartite.
is_connected	Specifies whether or not the input graph is connected.
is_cycle	Determines whether the input graph is a simple cycle.
is_dag	Determines whether the input graph is a directed acyclic graph (DAG).
is_directed	Specifies whether or not the graph is directed.
is_hamiltonian	Determines whether the graph is Hamiltonian (has a Hamiltonian cycle) or not.
is_immutable	Specifies whether or not the graph is immutable.
is_multigraph	Specifies whether or not the graph allows multiedges.
is_path	Determines whether the input graph is a simple path.

Method Name	Description
is_strongly_connected	Specifies whether the input graph is strongly connected.
is_weakly_connected	Specifies whether the input graph is weakly connected.
label_propagation	Computes the communities of the graph by using the label propagation algorithm.
label_propagation_nodes	Computes community assignments for all nodes in the graph by using the label propagation algorithm.
leaf_node_count	Computes a graph's leaf node count.
local_transitivity	Computes the local transitivity (clustering coefficient) and stores the results as node attributes on the graph.
louvain	Computes the communities of the graph by using the Louvain algorithm.
louvain_nodes	Computes community assignments for all nodes in the graph by using the Louvain algorithm.
maxflow	Calculates the maximum flow from a source node to a sink node in a graph.
maximal_cliques	Enumerates all maximal cliques of the graph.
min_cut	Calculates a minimum cut of a graph.
min_s_t_cut	Calculates a minimum s-t cut of a graph.
mincostflow	Calculates the minimum-cost network flow for the graph.
minimum_spanning_tree	Calculates a minimum spanning tree (MST) or forest of a graph.
minimum_weight_full_matching	Returns a minimum-weight full matching of the bipartite graph *self*.
neighbors	Returns an iterator over all successor nodes of node n.
node_clique_numbers	Computes node clique numbers and stores the results as attributes on the graph.
number_of_edges	Returns the number of edges in the graph.
number_of_nodes	Returns the number of nodes in the graph.

Method Name	Description
out_degree	Returns the out-degree of nodes.
predecessors	Returns an iterator over all predecessor nodes of node n.
projected_graph	Computes a projected graph.
projected_graph_adamic_adar	Computes an Adamic-Adar projected graph.
projected_graph_common_neighbors	Computes a common neighbors projected graph.
projected_graph_cosine	Computes a cosine projected graph.
projected_graph_dice	Computes a Sørensen-Dice projected graph.
projected_graph_jaccard	Computes a Jaccard projected graph.
query	Queries for the subgraph isomorphisms of q within g.
reach_graph	Computes the reach graph for the source or set of sources.
reach_graph_per_source	Computes a separate reach graph for each given source node.
set_edge_attributes	Sets edge attributes by using a given value or dictionary of values.
set_node_attributes	Sets node attributes from a given value or dictionary of values.
shortest_path_distances	Calculates the shortest path distances in a graph.
similarity_adamic_adar	Computes Adamic-Adar node similarity.
similarity_common_neighbors	Computes common neighbors node similarity.
similarity_cosine	Computes cosine node similarity.
similarity_dice	Computes Sørensen-Dice node similarity.
similarity_graph	Computes a node similarity graph.
similarity_graph_adamic_adar	Computes an Adamic-Adar node similarity graph.
similarity_graph_common_neighbors	Computes a common neighbors node similarity graph.

Method Name	Description
similarity_graph_cosine	Computes a cosine node similarity graph.
similarity_graph_dice	Computes a Sørensen-Dice node similarity graph.
similarity_graph_jaccard	Computes a Jaccard node similarity graph.
similarity_jaccard	Computes Jaccard node similarity.
singleton_node_count	Computes a graph's singleton node count.
successors	Returns an iterator over all successor nodes of node n.
to_directed	Returns a directed copy of the input graph.
to_immutable	Creates an immutable graph from the input graph.
to_mutable	Creates a mutable graph from the input graph.
to_pandas_edges	Returns the graph edges as a Pandas DataFrame.
to_pandas_nodes	Returns the graph nodes as a Pandas DataFrame.
to_undirected	Returns a directed copy of the input graph.
topological_ordering	Finds a topological ordering of the graph's nodes.
transitive_closure	Calculates the transitive closure of a graph.
triangle_count	Computes a graph's triangle count.
tsp_tour	Calculates a traveling salesman problem (TSP) tour of a graph.
union	Returns the union of input graphs by finding the unions of their nodes and edges.
vrp	Solves the vehicle routing problem.
vrptw	Solves the vehicle routing problem with time windows.

add_core_number_attr

```
add_core_number_attr(maxtime=None, *, inplace=False, data=True)
```

Computes the core numbers and stores the results as node attributes on the graph.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time for core decomposition to spend. The default is None.

inplace : bool, optional
Specifies whether to add the core numbers as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph with core numbers that are stored as a node attribute named *core*. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

add_topological_order_attr

```
add_topological_order_attr(*, inplace=False, data=True)
```

Computes a topological ordering and stores the results as node attributes of the graph.

Parameters

inplace : bool, optional
Specifies whether or not to add the topological order values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a directed graph whose topological order is stored as a node attribute named *top_order*. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

adjacency

```
adjacency()
```

Returns an iterator over (node, adjacency dict) tuples for all nodes.

Returns

iterator
Returns an iterator of (node,adj dict) for all nodes.

Examples

```
G = sasnet.DiGraph([[1, 2, {'h': 2}],
                    [1, 3, {'w': 2, 'h': 0}],
                    [2, 3, {'w': 3, 'h': 5}]])
G_out = sasnet.ImmutableDiGraph(G)
print(list(G_out.adjacency()))
[(1, {2: {'h': 2}, 3: {'w': 2, 'h': 0}}), (2, {3: {'w': 3, 'h': 5}}), (3, {})].
```

allows_multiedges

```
allows_multiedges()
```

Specifies whether or not the graph allows multiedges.

Returns

bool
Indicates whether or not the graph allows multiedges.

allows_selfloops

```
allows_selfloops()
```

Specifies whether or not the graph allows self-loops.

Returns

bool
Indicates whether or not the graph allows self-loops.

approximate_diameter

```
approximate_diameter(edge_weight=None)
```

Computes an approximation of the graph's diameter.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted approximate diameter is computed.

Returns

float
Returns the approximate diameter.

articulation_point_count

```
articulation_point_count()
```

Computes a graph's articulation point count.

Returns

int
Returns the number of articulation points.

articulation_points

```
articulation_points()
```

Finds the articulation points (cut points) of a graph.

Returns

list
Returns a list of articulation points in the graph.

assortativity_degree

```
assortativity_degree(source_direction='out', target_direction='in',
edge_weight=None)
```

Computes a graph's degree assortativity.

Parameters

source_direction : {'in', 'out'}
Specifies the degree type (in-degree or out-degree) for the source node in a directed graph.

target_direction : {'in', 'out'}
Specifies the degree type (in-degree or out-degree) for the target node in a directed graph.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted assortativity degree is computed.

Returns

float
Returns the degree assortativity.

assortativity_nominal

```
assortativity_nominal(node_attr)
```

Computes a graph's nominal assortativity.

Parameters

node_attr : str
Specifies the name of the nominal node attribute to use.

Returns

float
Returns the nominal assortativity.

assortativity_numeric

```
assortativity_numeric(node_attr)
```

Computes a graph's numeric assortativity.

Parameters

node_attr : str
Specifies the name of the numeric node attribute to use.

Returns

float
Returns the numeric assortativity.

average_shortest_path

```
average_shortest_path(edge_weight=None)
```

Computes a graph's average shortest path.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted average shortest path is computed.

Returns

float
Returns the average shortest path.

biconcomp_distribution_edges

```
biconcomp_distribution_edges(bins=None)
```

Computes the distribution of the number of edges in each biconnected components by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is int (an integer), it specifies the number of equal-width bins. When *bins* is a list, it specifies bin ranges and should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is str (a string), it specifies the method to use for calculating the bin ranges. For example, 'stone','auto','doane','fd','rice','scott','sqrt','sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of biconnected components whose edge count is within the range of the bin.

biconcomp_distribution_nodes

```
biconcomp_distribution_nodes(bins=None)
```

Computes the distribution of the number of nodes in each biconnected components by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is int (an integer), it specifies the number of equal-width bins. When *bins* is a list, it specifies bin ranges and should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is str (a string), it specifies the method to use for calculating the bin ranges. For example, 'stone','auto','doane','fd','rice','scott','sqrt','sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of biconnected components whose node count is within the range of the bin.

biconcomp_size_edges

```
biconcomp_size_edges()
```

Computes the number of edges within each biconnected component.

Returns

dict
Returns a dictionary in which the keys are biconnected component IDs and the values are the number of edges within each component.

biconcomp_size_nodes

```
biconcomp_size_nodes()
```

Computes the number of nodes within each biconnected component.

Returns

dict
Returns a dictionary in which the keys are biconnected component IDs and the values are the number of nodes within each component.

biconnected_component_count

```
biconnected_component_count()
```

Computes a graph's biconnected component count.

Returns

int
Returns the number of biconnected components.

biconnected_components

```
biconnected_components(order_by=None, *, data=True, ascending=False)
```

Computes the graph's biconnected components for an undirected graph.

Parameters

order_by : 'node' or 'edge' or None, optional
Specifies the parameter that is used to sort the components by the number of nodes or edges within each component. The default is None, which means that the components are not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Returns the result as a generator of biconnected component subgraphs. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

block_cut_tree

```
block_cut_tree(*, return_nodes_map=False)
```

Computes the block-cut tree of a graph.

Parameters

return_nodes_map : bool, optional
Specifies whether or not to also compute the block-cut tree node map. The default is False.

Returns

SAS graph object
Returns a graph that represents the block-cut tree. When *return_nodes_map = True*, the output graph has the *nodes_map* attribute, which contains a dictionary that maps its nodes to a list of nodes in the original graph. Note: If a block is empty (that is, the corresponding biconnected component contains only articulation points), that block does not appear as a key in the *nodes_map* dictionary.

centrality_betweenness

```
centrality_betweenness(edge_weight=None, sample_percent=100, *,
inplace=False, data=True)
```

Computes weighted or unweighted betweenness centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that unweighted betweenness centrality is computed.

sample_percent : int, optional
Specifies the percentage of source nodes to sample for the approximate betweenness calculation. This parameter value should be in the range (0, 100]. The default is 100.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the node and edge attributes *between_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_closeness

```
centrality_closeness(edge_weight=None, nopath='diameter', *, inplace=False,
data=True)
```

Computes weighted or unweighted closeness centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted closeness centrality is computed.

nopath : {'diameter', 'harmonic', or 'zero'}, optional
Specifies a method to use for accounting for the shortest path distance between two nodes when a path does not exist. The valid values are as follows: - 'diameter' uses the graph diameter (plus one) as the shortest path distance between disconnected nodes. - 'harmonic' uses the harmonic formula for calculating closeness centrality. - 'zero' uses zero as the shortest path distance between disconnected nodes.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the closeness centrality node attributes *centr_close_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_degree

```
centrality_degree(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted degree centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted degree centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the degree centrality node attributes *centr_degree_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_eigenvector

```
centrality_eigenvector(edge_weight=None, maxiters=10000, *, inplace=False,
data=True)
```

Computes weighted or unweighted eigenvector centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted eigenvector centrality is computed.

maxiters : int, optional
Specifies the maximum number of iterations in order to limit the amount of computation time that is spent when convergence is slow. The default is 10,000.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the degree centrality node attributes *centr_eigen_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_hub_authority

```
centrality_hub_authority(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted hub and authority centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted hub and authority centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the hub and authority centrality node attributes *centr_hub_** and *centr_auth_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_influence

```
centrality_influence(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted influence centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted influence centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the first- and second-order influence centrality node attributes *centr_influence1_** and *centr_influence2_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_pagerank

```
centrality_pagerank(edge_weight=None, alpha=0.85, tolerance=1e-09, *,
inplace=False, data=True)
```

Computes weighted or unweighted PageRank centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted PageRank centrality is computed.

alpha : float, optional
Specifies the damping parameter for the PageRank centrality (or algorithm). The default is 0.85.

tolerance : float, optional
Specifies the tolerance parameter for the PageRank centrality (or algorithm). The default is 1E-9.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the PageRank centrality node attributes *centr_pagerank_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

concomp_distribution_edges

```
concomp_distribution_edges(bins=None, *, strongly=True)
```

Computes the distribution of the number of edges in each connected component by using *numpy.histogram*.

Parameters

bins : int or list or str, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is an integer (int), it represents the number of equal-width bins. When *bins* is a list, it specifies bin ranges and it should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is a string (str), it specifies the method to be used for calculating the bin ranges. For example, 'stone', 'auto', 'doane', 'fd', 'rice', 'scott', 'sqrt', 'sturges'. See https://

numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of components whose edge count within the range of the bin.

concomp_distribution_nodes

```
concomp_distribution_nodes(bins=None, *, strongly=True)
```

Computes the distribution of the number of nodes in each connected component by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is an integer (int), it represents the number of equal-width bins. When *bins* is a list, it specifies bin ranges and it should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is a string (str), it specifies the method to use for calculating the bin ranges. For example, 'stone', 'auto', 'doane', 'fd', 'rice', 'scott', 'sqrt', 'sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of components whose node count is within the range of the bin.

concomp_size_edges

```
concomp_size_edges(*, strongly=True)
```

Computes the number of edges within each connected component.

Parameters

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary in which the keys are connected component IDs and the values are the number of edges within each component.

concomp_size_nodes

```
concomp_size_nodes(*, strongly=True)
```

Computes the number of nodes within each connected component.

Parameters

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary in which the keys are connected component IDs and the values are the number of nodes within each component.

connected_component_count

```
connected_component_count()
```

Computes a graph's connected component count.

Returns

int
Returns the number of connected components.

connected_components

```
connected_components(order_by=None, *, strongly=True, data=True,
ascending=False)
```

Computes the connected components of a graph.

Parameters

order_by : 'node' or 'edge' or None, optional
Specifies whether to sort the components by the number of nodes or edges within each component. The default is None, which means that the components are not sorted.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one connected component. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

copy

```
copy(immutable=None, *, data=True)
```

Returns a copy of the input graph.

Parameters

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, meaning that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns an independent copy of the input.

core_decomposition

```
core_decomposition(maxtime=None, *, data=True)
```

Computes a core decomposition of the graph.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time for core decomposition to spend. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one k-core in increasing core number. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

cycle_count

```
cycle_count(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Counts the number of cycles in a graph.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Returns

int
Returns a count of cycles in the graph.

degree

```
degree(n=None, edge_weight=None)
```

Returns the degree of nodes.

Parameters

n : single node or container of nodes or None, optional
Specifies which nodes to include. The default is None, which means that the degree of all nodes is returned.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None.

Returns

int or dictionary
Returns the degree of specified node(s).

density

```
density()
```

Computes the density of the graph.

Returns

float
Returns the density of the graph.

diameter

```
diameter(edge_weight=None)
```

Computes a graph's diameter.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted diameter is computed.

Returns

int or float
Returns the diameter.

drop_edge_attributes

```
drop_edge_attributes(attributes_subset=None, edges_subset=None,
immutable=None, *, inplace=False)
```

Removes edge attributes if they exist.

Parameters

attributes_subset : str or list, optional
Specifies the edge attribute or a list of the edge attributes to remove; for example, 'weight' or ['weight','color']. By default, all edge attributes are removed.

edges_subset : list or iterable of edges, optional
Specifies the edges to remove attributes from; for example, [(1,2),(2,3)]. By default the attributes for all edges are removed.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by removing the specified edge attributes. If the *inplace* parameter value is False, a copy of the input graph with the modified edge attributes is returned.

drop_node_attributes

```
drop_node_attributes(attributes_subset=None, nodes_subset=None,
immutable=None, *, inplace=False)
```

Removes node attributes if they exist.

Parameters

attributes_subset : str or list, optional
Specifies the node attribute or list of node attributes to remove; for example, ['weight','color']. By default, all node attributes are removed.

nodes_subset : list or iterable of nodes, optional
Specifies the nodes to drop attributes from; for example, [1,2,3] or (1,2,3) or [1] or (1,). By default, the attributes for all nodes are removed.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by removing the specified node attributes. If the *inplace* parameter value is False, a copy of the input graph with the modified node attributes is returned.

enumerate_cycle_edges

```
enumerate_cycle_edges(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Enumerates the cycles in a graph and returns the edges of the cycles.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Yields

generator
Yields a generator of cycles, each of which is given as a list of the cycles' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_cycle_nodes

```
enumerate_cycle_nodes(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Enumerates the cycles in a graph and returns the nodes of the cycles.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Yields

generator
Yields a generator of cycles, each of which is given as a list of the cycles' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_cycles

```
enumerate_cycles(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None, *,
data=True)
```

Enumerates the cycles in a graph and returns the cycle graph objects.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one cycle of the same type as the input graph. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_path_edges

```
enumerate_path_edges(source=None, sink=None, minlength=1,
maxlength=None, edge_weight=None, node_weight=None, minedgewt=None,
maxedgewt=None, minnodewt=None, maxnodewt=None, maxtime=None)
```

Enumerates the paths in a graph and returns the edges of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

Yields

generator
Yields a generator of Path objects, each of which is given as a list of the paths' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_path_nodes

```
enumerate_path_nodes(source=None, sink=None, minlength=1,
maxlength=None, edge_weight=None, node_weight=None, minedgewt=None,
maxedgewt=None, minnodewt=None, maxnodewt=None, maxtime=None)
```

Enumerates the paths in a graph and returns the nodes of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

Yields

generator
Yields a generator of Path objects, each of which is given as a list of the paths' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_paths

```
enumerate_paths(source=None, sink=None, minlength=1, maxlength=None,
edge_weight=None, node_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, *, data=True)
```

Enumerates the paths in a graph and returns the Path graph objects.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one Path object. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_sequence_shortest_paths

```
enumerate_sequence_shortest_paths(sequence, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None, *, data=True)
```

Enumerates the shortest paths in a graph that visit the indicated sequence nodes in order.

Parameters

sequence : iterable of nodes
Specifies the order in which to visit the required nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Path objects; each object represents a subpath between a sequence of nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_path_edges

```
enumerate_shortest_path_edges(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None)
```

Enumerates the shortest paths in a graph and returns the edges of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

Yields

generator
Yields a generator of paths; each path is given as a list of the paths' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_path_nodes

```
enumerate_shortest_path_nodes(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None)
```

Enumerates the shortest paths in a graph and returns the nodes of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

Yields

generator
Yields a generator of paths; each path is given as a list of the paths' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_paths

```
enumerate_shortest_paths(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None, *, data=True)
```

Enumerates the shortest paths in a graph and returns the Path graph objects.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Path objects; each object represents one shortest path. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

get_meta_graph

```
get_meta_graph(node_type, edge_type=None, immutable=None)
```

Constructs a metagraph from the input graph.

Parameters

node_type : str
Specifies the node attribute to consider in constructing the metagraph.

edge_type : str, optional
Specifies the edge attribute to consider in constructing the metagraph. By default, all connections between nodes are considered the same.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

Returns

SAS graph object
Returns the metagraph, which has an edge attribute count and determines the number of unique connections between pairs of nodes.

get_nlargest_biconcomps

```
get_nlargest_biconcomps(n, order_by='node', *, data=True, ascending=False)
```

Returns a list of up to *n* largest biconnected components.

Parameters

n : int
Specifies the number of graphs to return.

order_by : 'node' or 'edge' or None
Specifies whether to sort the biconnected components by the number of nodes or edges within those components. The default is None, which means that the biconnected components are not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. When the value is True, the smallest biconnected components are returned instead. The default is False.

Returns

list
Returns a list of n largest or smallest biconnected components of a graph.

get_nlargest_concomps

```
get_nlargest_concomps(n, order_by='node', *, strongly=True, data=True,
ascending=False)
```

Returns a list of up to *n* largest connected (with highest number of nodes) components of a graph.

Parameters

n : int
Specifies the number of graphs to return.

order_by : 'node' or 'edge'
Specifies whether to sort the component by number of nodes or edges within that component. The default is None, which means that the result is not sorted.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Returns

list
Returns a list of the n largest connected components of a graph.

has_edge_attribute

```
has_edge_attribute(name, *, all_edges=False)
```

Checks the edges of the input graph for the specified edge attribute.

Parameters

name : str
Specifies the name of the edge attribute.

all_edges : bool, optional
Specifies whether or not to check for the existence of the edge attribute in all or any of the edges. The default is False.

Returns

bool
If the all_edges parameter value is True, the value True is returned if that edge attribute exists in all edges; otherwise the value False is returned. If the all_edges parameter value is False, the value True is returned if at least one edge in the graph has that specified edge attribute.

has_node_attribute

```
has_node_attribute(name, *, all_nodes=False)
```

Checks the nodes of the input graph for the specified node attribute.

Parameters

name : str
Specifies the name of the node attribute.

all_nodes : bool, optional
Specifies whether or not to check for the existence of the node attribute in all or any of the nodes. The default is False.

Returns

bool
If the all_nodes parameter value is True, the value True is returned if that node attribute exists in all node; otherwise the value False is returned. If the all_nodes parameter value is False, the value True is returned if at least one node in the graph has that specified node attribute.

in_degree

```
in_degree(n=None, edge_weight=None)
```

Returns the in-degree of nodes.

Parameters

n : single node or container of nodes or None, optional
Specifies which nodes to include. The default is None, which means that the in-degree of all nodes is returned.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None.

Returns

numeric or dictionary
Returns the in-degree of the node if a single node is passed, or returns a dictionary with nodes as keys and their in-degrees as values if None or a list of nodes is passed.

intersection

```
intersection(rhs, nodes_attrs_agg=None, edges_attrs_agg=None)
```

Returns the intersection of input graphs by finding the intersections of their nodes and edges.

The node attributes and edge attributes are aggregated using the rules that are defined by the *nodes_attrs_agg* and *edges_attrs_agg* parameters.

Parameters

rhs : SAS graph object
Specifies the other graph, of the same type as *self*, with which to intersect.

nodes_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate node attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps node attributes to functions. The default is None.

edges_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate edge attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps edge attributes to functions. The default is None.

Returns

SAS graph object
Returns a graph that is constructed by the intersection of the two graphs.

is_biconnected

```
is_biconnected()
```

Specifies whether or not the input graph is biconnected.

Returns

bool
Indicates whether the input graph is biconnected (True) or not (False).

is_bipartite

```
is_bipartite()
```

Determines whether the input graph is bipartite.

Returns

bool
Indicates whether the input graph is bipartite.

is_connected

```
is_connected()
```

Specifies whether or not the input graph is connected.

Returns

bool
Indicates whether the input graph is connected (True) or not (False).

is_cycle

```
is_cycle()
```

Determines whether the input graph is a simple cycle.

Returns

bool
Indicates whether the input graph is a simple cycle (True) or not (False).

is_dag

```
is_dag()
```

Determines whether the input graph is a directed acyclic graph (DAG).

Returns

bool
Indicates whether the input graph is a directed acyclic graph (DAG).

is_directed

```
is_directed()
```

Specifies whether or not the graph is directed.

Returns

bool
Indicates whether or not the graph is directed.

is_hamiltonian

```
is_hamiltonian(maxtime=None)
```

Determines whether the graph is Hamiltonian (has a Hamiltonian cycle) or not.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time to determine whether or not the graph is Hamiltonian. The default is None.

Returns

bool or None
Indicates whether or not the input graph is determined to be Hamiltonian. Returns None if no determination is made.

is_immutable

```
is_immutable()
```

Specifies whether or not the graph is immutable.

Returns

bool
Indicates whether or not the graph is immutable.

is_multigraph

```
is_multigraph()
```

Specifies whether or not the graph allows multiedges.

Returns

bool
Indicates whether or not the graph allows multiedges.

is_path

```
is_path()
```

Determines whether the input graph is a simple path.

Returns

bool
Indicates whether the input graph is a simple path (True) or not (False).

is_strongly_connected

```
is_strongly_connected()
```

Specifies whether the input graph is strongly connected.

Returns

bool
Indicates whether the input graph is strongly connected (True) or not (False).

is_weakly_connected

```
is_weakly_connected()
```

Specifies whether the input graph is weakly connected.

Returns

bool
Indicates whether the input graph is weakly connected (True) or not (False).

label_propagation

```
label_propagation(edge_weight=None, maxiters=100, tolerance=0.05, *,
data=True)
```

Computes the communities of the graph by using the label propagation algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of label changes for all nodes in the graph is less than the value. The default is 0.05.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one community. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

label_propagation_nodes

```
label_propagation_nodes(edge_weight=None, maxiters=100, tolerance=0.05, *,
data=True, inplace=False)
```

Computes community assignments for all nodes in the graph by using the label propagation algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of label changes for all nodes in the graph is less than the value. The default is 0.05.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

inplace : bool, optional
Specifies whether to add the community values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains the node attribute *community*, which denotes the community assignment for each node. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

leaf_node_count

```
leaf_node_count()
```

Computes a graph's leaf node count.

Returns

int
Returns the number of leaf nodes.

local_transitivity

```
local_transitivity(*, inplace=False)
```

Computes the local transitivity (clustering coefficient) and stores the results as node attributes on the graph.

Parameters

inplace : bool, optional
Specifies whether to add the transitivity values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains node attribute 'transitivity'. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

louvain

```
louvain(*, edge_weight=None, resolution=1, tolerance=0.001, maxiters=100,
data=True)
```

Computes the communities of the graph by using the Louvain algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

resolution : float, optional
Specifies the factor in the modularity equation that affects the expected number of links between two nodes. A higher value produces more communities. The default is 1.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of modularity gain between two consecutive iterations is less than the value. The default is 0.001.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one community. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

louvain_nodes

```
louvain_nodes(*, edge_weight=None, resolution=1, tolerance=0.001,
maxiters=100, data=True, inplace=False)
```

Computes community assignments for all nodes in the graph by using the Louvain algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

resolution : float, optional
Specifies the factor in the modularity equation that affects the expected number of links between two nodes. A higher value produces more communities. The default is 1.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of modularity gain between two consecutive iterations is less than the value. The default is 0.001.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

inplace : bool, optional
Specifies whether to add the community values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains the node attribute *community*, which denotes the community assignment for each node. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

maxflow

```
maxflow(source, sink, capacity_attr='upper', *, inplace=False, data=True)
```

Calculates the maximum flow from a source node to a sink node in a graph.

Parameters

source : number or str
Specifies the source node for the maximum flow.

sink : number or str
Specifies the sink node for the maximum flow.

capacity_attr : str, optional
Specifies the name of the edge attribute that contains the capacity of the edge. The default is 'upper'.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute *flow*, which contains the flow values. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, the returned graph contains only the flow edges.

maximal_cliques

```
maximal_cliques(maxcliques='ALL', node_weight=None, edge_weight=None,
minsize=1, maxsize=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, order_by=None, *,
data=True, ascending=False)
```

Enumerates all maximal cliques of the graph.

Parameters

maxcliques : int or 'ALL', optional
Specifies the maximum number of enumerated cliques to return. The default is 'ALL'.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

minsize : int, optional
Specifies the minimum number of nodes in a clique. The default is 1.

maxsize : int or None, optional
Specifies the maximum number of nodes in a clique. The default is None.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a clique. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a clique. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a clique. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a clique. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds for enumeration to spend. The default is None.

order_by : 'node' or None, optional
Specifies whether to sort the cliques by the number of nodes within each clique. The default is None, which means that the result is not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one maximal clique. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

min_cut

```
min_cut(edge_weight=None, *, inplace=False, data=True)
```

Calculates a minimum cut of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted minimum cut is calculated.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute 'mincut', which indicates the edges to include in the minimum cut set, and the node attribute 'mincut_partition', which indicates the node partition of the calculated minimum cut. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, a modified copy of the graph is returned.

min_s_t_cut

```
min_s_t_cut(source, sink, edge_weight=None, *, inplace=False, data=True)
```

Calculates a minimum s-t cut of a graph.

Parameters

source : number or str
Specifies the source node (s) for the minimum s-t cut.

sink : number or str
Specifies the sink node (t) for the minimum s-t cut.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted minimum s-t cut is calculated.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute *s_t_cut*, which indicates the edges to include in the minimum s-t cut set; and the node attribute *s_t_cut_partition*, which indicates the node partition of the calculated minimum s-t cut. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, a modified copy of the graph is returned.

mincostflow

```
mincostflow(demand_lower_attr='lower', demand_upper_attr='upper',
capacity_lower_attr='lower', capacity_upper_attr='upper', cost_attr='weight',
maxtime=None, *, inplace=False, data=True)
```

Calculates the minimum-cost network flow for the graph.

Parameters

demand_lower_attr : str, optional
Specifies the name of the node attribute that contains the lower bound of the node supply. The default is 'lower'.

demand_upper_attr : str, optional
Specifies the name of the node attribute that contains the upper bound of the node supply. The default is 'upper'.

capacity_lower_attr : str, optional
Specifies the name of the edge attribute that contains the capacity lower bound of the edge. The default is 'lower'.

capacity_upper_attr : str, optional
Specifies the name of the edge attribute that contains the capacity upper bound of the edge. The default is 'upper'.

cost_attr : str, optional
Specifies the name of the edge attribute that contains the edge cost. The default is 'weight'.

maxtime : float or None, optional
Specifies the maximum amount of time to allow for computing the minimum-cost network flow. The default is None.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attributes *flow* and *reduced_cost*, which contain the flow and reduced cost values, respectively; and the node attribute *dual*, which contains the optimal dual value. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned.

minimum_spanning_tree

```
minimum_spanning_tree(edge_weight=None, source=None, *, inplace=False,
data=True)
```

Calculates a minimum spanning tree (MST) or forest of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

source : number or str or None
Specifies the source node for a directed MST. The default is None.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute 'mst', which indicates the edges to include in the spanning tree or forest. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, the returned graph contains only the spanning tree or forest edges.

minimum_weight_full_matching

```
minimum_weight_full_matching(edge_weight=None, *, data=True)
```

Returns a minimum-weight full matching of the bipartite graph *self*.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns a directed graph whose edges identify the assignment mapping in the minimum-weight full matching solution. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

neighbors

```
neighbors(n)
```

Returns an iterator over all successor nodes of node n.

Parameters

n : node
Specifies a node in the graph.

Returns

iterator
Returns an iterator over all successor nodes of node n.

node_clique_numbers

```
node_clique_numbers(*, inplace=False, data=True)
```

Computes node clique numbers and stores the results as attributes on the graph.

Parameters

inplace : bool, optional
Specifies whether to add the clique number values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the value of the *inplace* parameter is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph or modifies the existing graph to contain the node attribute *clique_number*, which includes the clique number of each node. The graph g has the attribute g.solution_summary, which contains information about the results of the algorithm.

number_of_edges

```
number_of_edges(u=None, v=None)
```

Returns the number of edges in the graph.

When you specify *u* and *v*, the number of edges between *u* and *v* is returned. When you specify *u* but *v* is None, the number of connections from *u* to other nodes (out-degree of node *u*) is returned. When you specify *v* but *u* is None, the number of connections from other nodes to *v* (in-degree of node *v*) is returned.

Parameters

u : numeric or str or None, optional
Specifies the *from* node. The default is None.

v : numeric or str or None, optional
Specifies the *to* node. The default is None.

Returns

int
Returns the number of edges in the graph. When you specify *u* and *v*, the number of edges between *u* and *v* is returned. When you specify *u* but *v* is None, the number of connections from *u* to other nodes (out-degree of node *u*) is returned. When you specify *v* but *u* is None, the number of connections from other nodes to *v* (in-degree of node *v*) is returned.

number_of_nodes

```
number_of_nodes()
```

Returns the number of nodes in the graph.

Returns

int
Returns the number of nodes in the graph.

out_degree

```
out_degree(n=None, edge_weight=None)
```

Returns the out-degree of nodes.

Parameters

n : single node or container of nodes or None, optional
Specifies which nodes to include. The default is None, which means that the out-degree of all nodes is returned.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None.

Returns

numeric or dictionary
Returns the out-degree of the node if a single node is passed, or returns a dictionary with nodes as keys and their out-degrees as values if None or a list of nodes is passed.

predecessors

```
predecessors(n)
```

Returns an iterator over all predecessor nodes of node n.

Parameters

n : node
Specifies a node in the graph.

Returns

iterator
Returns an iterator over all predecessor nodes of node n.

projected_graph

```
projected_graph(nodes, neighbors=None, directed_method=None,
_measure=None, *, multigraph=False, data=True)
```

Computes a projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in self other than *nodes*.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used. When the *directed_method* parameter is specified for an undirected graph, a ValueError is raised.

multigraph : bool, optional
Specifies whether or not to return a multigraph. When the value is True, the algorithm returns a multigraph in which the multiple edges represent multiple shared neighbors. The edge attribute 'neighbor' stores the label of the shared neighbor. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

projected_graph_adamic_adar

```
projected_graph_adamic_adar(nodes, neighbors=None, directed_method=None,
*, data=True)
```

Computes an Adamic-Adar projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than *nodes*.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'adamic_adar' edge attribute.

projected_graph_common_neighbors

```
projected_graph_common_neighbors(nodes, neighbors=None,
directed_method=None, *, data=True)
```

Computes a common neighbors projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'common_neighbors' edge attribute.

projected_graph_cosine

```
projected_graph_cosine(nodes, edge_weight=None, neighbors=None,
directed_method=None, *, data=True)
```

Computes a cosine projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'cosine' edge attribute.

projected_graph_dice

```
projected_graph_dice(nodes, neighbors=None, directed_method=None, *,
data=True)
```

Computes a Sørensen-Dice projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the subset of nodes that are specified by the *neighbors* parameter. The strength of association in the calculated projection is represented by the 'dice' edge attribute.

projected_graph_jaccard

```
projected_graph_jaccard(nodes, neighbors=None, directed_method=None, *,
data=True)
```

Computes a Jaccard projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'jaccard' edge attribute.

query

```
query(q, expr=None, _node_attr_to_ignore=None, _edge_attr_to_ignore=None, *,
induced=False, break_symmetry=False, _generate_output=True,
_orbits_only=False, **kwargs)
```

Queries for the subgraph isomorphisms of q within g.

Parameters

q : graph
Specifies the query graph.

expr : _Expression or None, optional
Specifies additional logic to filter the query results. The default is None.

induced : bool, optional
Specifies whether or not to return only node-induced matches. The default is False.

break_symmetry : bool, optional
Specifies whether or not to break symmetry and eliminate automorphic relationships between matches. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one isomorphic mapping from *q* to *self*. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

reach_graph

```
reach_graph(source, max_reach=1, *, data=True)
```

Computes the reach graph for the source or set of sources.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to compute the reach network.

max_reach : int, optional
Specifies the number of steps (or hops) to traverse from the source nodes. The default is 1.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns a graph that represents the reach network. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

reach_graph_per_source

```
reach_graph_per_source(source, max_reach=1, *, data=True)
```

Computes a separate reach graph for each given source node.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to compute the reach network.

max_reach : int, optional
Specifies the number of steps (or hops) to traverse from the source nodes. The default is 1.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one reach network. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

set_edge_attributes

```
set_edge_attributes(values, name=None, immutable=None, *, inplace=False)
```

Sets edge attributes by using a given value or dictionary of values.

Parameters

values : scalar or dict-like or Pandas DataFrame
Specifies what the edge attribute should be set to. When this parameter is set to a Pandas DataFrame, it is expected to have from, to, and edge_key (only for multigraphs) columns, and the other columns are treated as edge attributes to add. When *values* is a dictionary, the keys are edges and the values are either edge attribute values or dictionaries of attribute name-value pairs. For multigraphs, the edge tuples must be of the form (u, v, key). For simple graphs, the keys must be tuples of the form (u, v). When this parameter is not set to a dictionary, it is treated as a single attribute value that is then applied to every edge in *self*. The attribute name is specified by the *name* parameter.

name : str or None, optional
Specifies the name of the edge attribute to set if the *values* parameter is set to a scalar. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is False, a copy of the input graph with the new edge attributes is returned. If the *inplace* parameter value is True, the graph is modified by setting the specified edge attributes.

set_node_attributes

```
set_node_attributes(values, name=None, immutable=None, *, inplace=False)
```

Sets node attributes from a given value or dictionary of values.

Parameters

values : scalar or dict-like or Pandas DataFrame
Specifies what the node attribute should be set to. When *values* is not a dictionary, it is treated as a single attribute value that is then applied to every node in *self*. This means that if you provide a mutable object, such as a list, updates to that object are reflected in the node attribute for every node. The attribute name is specified by the *name* parameter. When *values* is a dictionary, the keys are nodes and the values are either node attribute values or dictionaries of attribute name-value pairs.

name : str or None, optional
Specifies the name of the node attribute to set if the *values* parameter is set to a scalar. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not the graph is modified in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by setting the specified node attributes. If the *inplace* parameter value is False, a copy of the input graph with the new node attributes is returned.

shortest_path_distances

```
shortest_path_distances(source=None, sink=None, edge_weight=None,
minedgewt=None, maxedgewt=None)
```

Calculates the shortest path distances in a graph.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) and restricts enumeration to the shortest paths that contain the specified source node(s). The default is None.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) and restricts enumeration to the shortest paths that contain the specified sink node(s). The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

Returns

dict
Returns a dictionary where each key is a (source, sink) tuple and its value is the corresponding shortest source-sink path distance.

similarity_adamic_adar

```
similarity_adamic_adar(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Adamic-Adar node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_common_neighbors

```
similarity_common_neighbors(source, sink=None, top_k=None, bottom_k=None,
*, exclude_source=False, exclude_existing_neighbors=False)
```

Computes common neighbors node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_cosine

```
similarity_cosine(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes cosine node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_dice

```
similarity_dice(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Sørensen-Dice node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_graph

```
similarity_graph(measure, min_score=2.220446049250313e-16, max_score=inf,
*, exclude_source=False, exclude_existing_neighbors=False, data=True,
**kwargs)
```

Computes a node similarity graph.

Parameters

measure : str, one of {'adamic_adar', 'common_neighbors', 'cosine', 'dice', 'jaccard'}
Specifies the metric to be used to compute the node similarity.

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the edge attribute that corresponds to the measure name. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_adamic_adar

```
similarity_graph_adamic_adar(min_score=2.220446049250313e-16,
max_score=inf, *, exclude_source=False, exclude_existing_neighbors=False,
data=True)
```

Computes an Adamic-Adar node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'adamic_adar' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_common_neighbors

```
similarity_graph_common_neighbors(min_score=2.220446049250313e-16,
max_score=inf, *, exclude_source=False, exclude_existing_neighbors=False,
data=True)
```

Computes a common neighbors node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'common_neighbors' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_cosine

```
similarity_graph_cosine(edge_weight=None,
min_score=2.220446049250313e-16, max_score=inf, *, exclude_source=False,
exclude_existing_neighbors=False, data=True)
```

Computes a cosine node similarity graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'cosine' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_dice

```
similarity_graph_dice(min_score=2.220446049250313e-16, max_score=inf, *,
exclude_source=False, exclude_existing_neighbors=False, data=True)
```

Computes a Sørensen-Dice node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the *dice* edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_jaccard

```
similarity_graph_jaccard(min_score=2.220446049250313e-16, max_score=inf, *,
exclude_source=False, exclude_existing_neighbors=False, data=True)
```

Computes a Jaccard node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'jaccard' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_jaccard

```
similarity_jaccard(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Jaccard node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

singleton_node_count

```
singleton_node_count()
```

Computes a graph's singleton node count.

Returns

int
Returns the number of singleton nodes.

successors

```
successors(n)
```

Returns an iterator over all successor nodes of node n.

Parameters

n : node
Specifies a node in the graph.

Returns

iterator
Returns an iterator over all successor nodes of node n.

to_directed

```
to_directed()
```

Returns a directed copy of the input graph.

Returns

SAS graph object
Returns a directed copy of the input graph.

to_immutable

```
to_immutable()
```

Creates an immutable graph from the input graph.

Returns

SAS graph object
Returns an immutable SAS graph object.

to_mutable

```
to_mutable()
```

Creates a mutable graph from the input graph.

Returns

SAS graph object
Returns a mutable SAS graph object.

to_pandas_edges

```
to_pandas_edges(*, data=True, _for_conversion_to_immutable=False)
```

Returns the graph edges as a Pandas DataFrame.

Parameters

data : bool or list, optional
Specifies whether or not to retain attributes. The default is True. When the value is True (or when a list is given), the Pandas DataFrame includes all the edge attributes (or all the listed edge attributes).

Returns

dataframe
Returns a Pandas DataFrame that includes the graph edges.

to_pandas_nodes

```
to_pandas_nodes(*, data=True, _for_conversion_to_immutable=False)
```

Returns the graph nodes as a Pandas DataFrame.

Parameters

data : bool or list, optional
Specifies whether or not to retain attributes. The default is True. When the value is True (or when a list is given), the Pandas DataFrame includes all the node attributes (or all the listed node attributes).

Returns

dataframe
Returns a Pandas DataFrame that includes the graph nodes.

to_undirected

```
to_undirected()
```

Returns a directed copy of the input graph.

Returns

SAS graph object
Returns a directed copy of the input graph.

topological_ordering

```
topological_ordering()
```

Finds a topological ordering of the graph's nodes.

Yields

generator
Yields a generator of nodes in topological order. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

transitive_closure

```
transitive_closure()
```

Calculates the transitive closure of a graph.

Returns

SAS graph object
Returns the transitive closure of the input graph that is returned as a SAS MultiGraph or MultiDiGraph object. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

triangle_count

```
triangle_count()
```

Computes a graph's triangle count.

Returns

int
Returns the number of triangles.

tsp_tour

```
tsp_tour(edge_weight=None, maxtime=None, minobjective=None, *,
inplace=False, use_milp=True, data=True, **options)
```

Calculates a traveling salesman problem (TSP) tour of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted TSP is calculated.

maxtime : float or None, optional
Specifies the maximum amount of time for solving a TSP. The default is None.

minobjective : float, optional
Specifies a minimum value such that when a solution is found whose objective is less than or equal to this value, the algorithm stops.

inplace : bool, optional
Specifies whether to add the TSP order values as attributes to the input graph directly or to create a copy of the graph. The default is False.

use_milp : bool, optional
Specifies whether or not to use a mixed integer linear programming (MILP) solver to solve a TSP. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is False, the returned graph is a Cycle that contains the calculated TSP tour edges. When the *inplace* parameter value is True, the graph is modified in place and None is returned. The modified graph contains the edge attributes *tsp* and *tsp_order* and the node attribute *tsp_order*.

union

```
union(rhs, nodes_attrs_agg=None, edges_attrs_agg=None)
```

Returns the union of input graphs by finding the unions of their nodes and edges.

The node attributes and edge attributes are aggregated using the rules that are defined by the *nodes_attrs_agg* and *edges_attrs_agg* parameters.

Parameters

rhs : SAS graph object
Specifies the other graph, of the same type as *self*, with which to find the union.

nodes_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate node attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps node attributes to functions. The default is None.

edges_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate edge attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps edge attributes to functions. The default is None.

Returns

SAS graph object
Returns a graph that is constructed by the union of the two graphs.

vrp

```
vrp(capacity, depot, node_demand='demand', edge_cost='weight', minroutes=1,
maxroutes=None, maxtime=None, minobjective=None, *, use_milp=True,
data=True, **options)
```

Solves the vehicle routing problem.

Parameters

capacity : float
Specifies the capacity of each vehicle.

depot : node
Specifies the depot node for the vehicle routing problem.

node_demand : str or dict or iterable
Specifies the quantity of goods to deliver to or pick up from a customer. The default is 'demand'.

edge_cost : str or dict or iterable
Specifies the cost of traversing each edge. The default is 'weight'.

minroutes : int, optional
Specifies the minimum number of routes that are allowed to service demand.

maxroutes : int, optional
Specifies the maximum number of routes that are allowed to service demand.

maxtime : float or None, optional
Specifies the maximum amount of time for solving the vehicle routing problem. The default is None.

minobjective : float, optional
Specifies a minimum value such that when a solution is found whose objective is less than or equal to this value, the algorithm stops.

use_milp : bool, optional
Specifies whether or not to use a mixed integer linear programming (MILP) solver to solve the vehicle routing problem. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Cycle objects; each cycle represents one route. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

vrptw

```
vrptw(capacity, depot, node_demand='demand', edge_cost='weight',
travel_time='weight', timewindow_lower=None, timewindow_upper=None,
service_time=None, minroutes=1, maxroutes=None, maxtime=None, *,
data=True)
```

Solves the vehicle routing problem with time windows.

Parameters

capacity : float
Specifies the capacity of each vehicle.

depot : node
Specifies the depot node for the vehicle routing problem.

node_demand : str or dict or iterable
Specifies the quantity of goods to deliver to or pick up from a customer. The default is 'demand'.

edge_cost : str or dict or iterable
Specifies the cost of traversing each edge.

travel_time : str or dict or iterable
Specifies attributes or values that define the time that it takes to traverse each edge. The default is 'weight'.

timewindow_lower : str or dict or iterable or None, optional
Specifies attributes or values that define the lower time limits within which a delivery vehicle can arrive at the customer nodes. The default is None.

timewindow_upper : str or dict or iterable or None, optional
Specifies attributes or values that define the upper time limits within which a delivery vehicle can arrive at the customer nodes. The default is None.

service_time : str or dict or iterable or None, optional
Specifies attributes or values that define the service time of the nodes. The default is None.

minroutes : int, optional
Specifies the minimum number of routes that are allowed to service demand.

maxroutes : int, optional
Specifies the maximum number of routes that are allowed to service demand.

maxtime : float or None, optional
Specifies the maximum amount of time for solving the vehicle routing problem. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Cycle objects; each cycle represents one route. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

ImmutableMultiGraph

Base class for undirected immutable multigraphs.

API: ImmutableMultiGraph

class
sasviya.network.classes.immutablemultigraph.ImmutableMultiGraph(graph=None, **kwargs)

Self-loops and multiedges are allowed.

Parameters

graph : sasnet.MultiGraph
Specifies the MultiGraph object to convert to an ImmutableMultiGraph object.

Attributes

edges
Returns the edges of the graph.

nodes
Returns the nodes of the graph.

Methods

Method Name	Description
add_core_number_attr	Computes the core numbers and stores the results as node attributes on the graph.
add_topological_order_attr	Computes a topological ordering and stores the results as node attributes of the graph.
adjacency	Returns an iterator over (node, adjacency dict) tuples for all nodes.
allows_multiedges	Specifies whether or not the graph allows multiedges.
allows_selfloops	Specifies whether or not the graph allows self-loops.
approximate_diameter	Computes an approximation of the graph's diameter.
articulation_point_count	Computes a graph's articulation point count.
articulation_points	Finds the articulation points (cut points) of a graph.
assortativity_degree	Computes a graph's degree assortativity.
assortativity_nominal	Computes a graph's nominal assortativity.
assortativity_numeric	Computes a graph's numeric assortativity.
average_shortest_path	Computes a graph's average shortest path.
biconcomp_distribution_edges	Computes the distribution of the number of edges in each biconnected components by using *numpy.histogram*.
biconcomp_distribution_nodes	Computes the distribution of the number of nodes in each biconnected components by using *numpy.histogram*.

Method Name	Description
biconcomp_size_edges	Computes the number of edges within each biconnected component.
biconcomp_size_nodes	Computes the number of nodes within each biconnected component.
biconnected_component_count	Computes a graph's biconnected component count.
biconnected_components	Computes the graph's biconnected components for an undirected graph.
block_cut_tree	Computes the block-cut tree of a graph.
centrality_betweenness	Computes weighted or unweighted betweenness centrality and stores the results as attributes on the graph.
centrality_closeness	Computes weighted or unweighted closeness centrality and stores the results as attributes on the graph.
centrality_degree	Computes weighted or unweighted degree centrality and stores the results as attributes on the graph.
centrality_eigenvector	Computes weighted or unweighted eigenvector centrality and stores the results as attributes on the graph.
centrality_hub_authority	Computes weighted or unweighted hub and authority centrality and stores the results as attributes on the graph.
centrality_influence	Computes weighted or unweighted influence centrality and stores the results as attributes on the graph.
centrality_pagerank	Computes weighted or unweighted PageRank centrality and stores the results as attributes on the graph.
concomp_distribution_edges	Computes the distribution of the number of edges in each connected component by using *numpy.histogram*.
concomp_distribution_nodes	Computes the distribution of the number of nodes in each connected component by using *numpy.histogram*.

Method Name	Description
concomp_size_edges	Computes the number of edges within each connected component.
concomp_size_nodes	Computes the number of nodes within each connected component.
connected_component_count	Computes a graph's connected component count.
connected_components	Computes the connected components of a graph.
copy	Returns a copy of the input graph.
core_decomposition	Computes a core decomposition of the graph.
cycle_count	Counts the number of cycles in a graph.
degree	Returns the degree of nodes.
density	Computes the density of the graph.
diameter	Computes a graph's diameter.
drop_edge_attributes	Removes edge attributes if they exist.
drop_node_attributes	Removes node attributes if they exist.
enumerate_cycle_edges	Enumerates the cycles in a graph and returns the edges of the cycles.
enumerate_cycle_nodes	Enumerates the cycles in a graph and returns the nodes of the cycles.
enumerate_cycles	Enumerates the cycles in a graph and returns the cycle graph objects.
enumerate_path_edges	Enumerates the paths in a graph and returns the edges of the paths.
enumerate_path_nodes	Enumerates the paths in a graph and returns the nodes of the paths.
enumerate_paths	Enumerates the paths in a graph and returns the Path graph objects.
enumerate_sequence_shortest_paths	Enumerates the shortest paths in a graph that visit the indicated sequence nodes in order.
enumerate_shortest_path_edges	Enumerates the shortest paths in a graph and returns the edges of the paths.

Method Name	Description
enumerate_shortest_path_nodes	Enumerates the shortest paths in a graph and returns the nodes of the paths.
enumerate_shortest_paths	Enumerates the shortest paths in a graph and returns the Path graph objects.
get_meta_graph	Constructs a metagraph from the input graph.
get_nlargest_biconcomps	Returns a list of up to *n* largest biconnected components.
get_nlargest_concomps	Returns a list of up to *n* largest connected (with highest number of nodes) components of a graph.
has_edge_attribute	Checks the edges of the input graph for the specified edge attribute.
has_node_attribute	Checks the nodes of the input graph for the specified node attribute.
intersection	Returns the intersection of input graphs by finding the intersections of their nodes and edges.
is_biconnected	Specifies whether or not the input graph is biconnected.
is_bipartite	Determines whether the input graph is bipartite.
is_connected	Specifies whether or not the input graph is connected.
is_cycle	Determines whether the input graph is a simple cycle.
is_dag	Determines whether the input graph is a directed acyclic graph (DAG).
is_directed	Specifies whether or not the graph is directed.
is_hamiltonian	Determines whether the graph is Hamiltonian (has a Hamiltonian cycle) or not.
is_immutable	Specifies whether or not the graph is immutable.
is_multigraph	Specifies whether or not the graph allows multiedges.
is_path	Determines whether the input graph is a simple path.
is_strongly_connected	Specifies whether the input graph is strongly connected.

Method Name	Description
is_weakly_connected	Specifies whether the input graph is weakly connected.
label_propagation	Computes the communities of the graph by using the label propagation algorithm.
label_propagation_nodes	Computes community assignments for all nodes in the graph by using the label propagation algorithm.
leaf_node_count	Computes a graph's leaf node count.
local_transitivity	Computes the local transitivity (clustering coefficient) and stores the results as node attributes on the graph.
louvain	Computes the communities of the graph by using the Louvain algorithm.
louvain_nodes	Computes community assignments for all nodes in the graph by using the Louvain algorithm.
maxflow	Calculates the maximum flow from a source node to a sink node in a graph.
maximal_cliques	Enumerates all maximal cliques of the graph.
min_cut	Calculates a minimum cut of a graph.
min_s_t_cut	Calculates a minimum s-t cut of a graph.
mincostflow	Calculates the minimum-cost network flow for the graph.
minimum_spanning_tree	Calculates a minimum spanning tree (MST) or forest of a graph.
minimum_weight_full_matching	Returns a minimum-weight full matching of the bipartite graph *self*.
neighbors	Returns an iterator over all neighbors of node n.
node_clique_numbers	Computes node clique numbers and stores the results as attributes on the graph.
number_of_edges	Returns the number of edges in the graph.
number_of_nodes	Returns the number of nodes in the graph.
projected_graph	Computes a projected graph.

Method Name	Description
projected_graph_adamic_adar	Computes an Adamic-Adar projected graph.
projected_graph_common_neighbors	Computes a common neighbors projected graph.
projected_graph_cosine	Computes a cosine projected graph.
projected_graph_dice	Computes a Sørensen-Dice projected graph.
projected_graph_jaccard	Computes a Jaccard projected graph.
query	Queries for the subgraph isomorphisms of q within g.
reach_graph	Computes the reach graph for the source or set of sources.
reach_graph_per_source	Computes a separate reach graph for each given source node.
set_edge_attributes	Sets edge attributes by using a given value or dictionary of values.
set_node_attributes	Sets node attributes from a given value or dictionary of values.
shortest_path_distances	Calculates the shortest path distances in a graph.
similarity_adamic_adar	Computes Adamic-Adar node similarity.
similarity_common_neighbors	Computes common neighbors node similarity.
similarity_cosine	Computes cosine node similarity.
similarity_dice	Computes Sørensen-Dice node similarity.
similarity_graph	Computes a node similarity graph.
similarity_graph_adamic_adar	Computes an Adamic-Adar node similarity graph.
similarity_graph_common_neighbors	Computes a common neighbors node similarity graph.
similarity_graph_cosine	Computes a cosine node similarity graph.
similarity_graph_dice	Computes a Sørensen-Dice node similarity graph.
similarity_graph_jaccard	Computes a Jaccard node similarity graph.

Method Name	Description
similarity_jaccard	Computes Jaccard node similarity.
singleton_node_count	Computes a graph's singleton node count.
to_directed	Returns a directed copy of the input graph.
to_immutable	Creates an immutable graph from the input graph.
to_mutable	Creates a mutable graph from the input graph.
to_pandas_edges	Returns the graph edges as a Pandas DataFrame.
to_pandas_nodes	Returns the graph nodes as a Pandas DataFrame.
to_undirected	Returns a directed copy of the input graph.
topological_ordering	Finds a topological ordering of the graph's nodes.
transitive_closure	Calculates the transitive closure of a graph.
triangle_count	Computes a graph's triangle count.
tsp_tour	Calculates a traveling salesman problem (TSP) tour of a graph.
union	Returns the union of input graphs by finding the unions of their nodes and edges.
vrp	Solves the vehicle routing problem.
vrptw	Solves the vehicle routing problem with time windows.

add_core_number_attr

```
add_core_number_attr(maxtime=None, *, inplace=False, data=True)
```

Computes the core numbers and stores the results as node attributes on the graph.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time for core decomposition to spend. The default is None.

inplace : bool, optional
Specifies whether to add the core numbers as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph with core numbers that are stored as a node attribute named *core*. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

add_topological_order_attr

```
add_topological_order_attr(*, inplace=False, data=True)
```

Computes a topological ordering and stores the results as node attributes of the graph.

Parameters

inplace : bool, optional
Specifies whether or not to add the topological order values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a directed graph whose topological order is stored as a node attribute named *top_order*. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

adjacency

```
adjacency()
```

Returns an iterator over (node, adjacency dict) tuples for all nodes.

Returns

iterator
Returns an iterator of (node,adj dict) for all nodes.

Examples

```
G = sasnet.MultiGraph([[1, 2, {'h': 2}],
                       [1, 2, {'w': 1}],
                       [1, 3, {'w': 2, 'h': 0}],
                       [2, 3, {'w': 3, 'h': 5}]])
G_out = sasnet.ImmutableMultiGraph(G)
print(list(G_out.adjacency()))
#[(1, {2: {0: {'h': 2}, 1: {'w': 1}}, 3: {0: {'w': 2, 'h': 0}}}),
# (2, {1: {0: {'h': 2}, 1: {'w': 1}}, 3: {0: {'w': 3, 'h': 5}}}),
# (3, {1: {0: {'w': 2, 'h': 0}}, 2: {0: {'w': 3, 'h': 5}}})].
```

allows_multiedges

```
allows_multiedges()
```

Specifies whether or not the graph allows multiedges.

Returns

bool
Indicates whether or not the graph allows multiedges.

allows_selfloops

```
allows_selfloops()
```

Specifies whether or not the graph allows self-loops.

Returns

bool
Indicates whether or not the graph allows self-loops.

approximate_diameter

```
approximate_diameter(edge_weight=None)
```

Computes an approximation of the graph's diameter.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted approximate diameter is computed.

Returns

float
Returns the approximate diameter.

articulation_point_count

```
articulation_point_count()
```

Computes a graph's articulation point count.

Returns

int
Returns the number of articulation points.

articulation_points

```
articulation_points()
```

Finds the articulation points (cut points) of a graph.

Returns

list
Returns a list of articulation points in the graph.

assortativity_degree

```
assortativity_degree(source_direction='out', target_direction='in',
edge_weight=None)
```

Computes a graph's degree assortativity.

Parameters

source_direction : {'in', 'out'}
Specifies the degree type (in-degree or out-degree) for the source node in a directed graph.

target_direction : {'in', 'out'}
Specifies the degree type (in-degree or out-degree) for the target node in a directed graph.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted assortativity degree is computed.

Returns

float
Returns the degree assortativity.

assortativity_nominal

```
assortativity_nominal(node_attr)
```

Computes a graph's nominal assortativity.

Parameters

node_attr : str
Specifies the name of the nominal node attribute to use.

Returns

float
Returns the nominal assortativity.

assortativity_numeric

```
assortativity_numeric(node_attr)
```

Computes a graph's numeric assortativity.

Parameters

node_attr : str
Specifies the name of the numeric node attribute to use.

Returns

float
Returns the numeric assortativity.

average_shortest_path

```
average_shortest_path(edge_weight=None)
```

Computes a graph's average shortest path.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted average shortest path is computed.

Returns

float
Returns the average shortest path.

biconcomp_distribution_edges

```
biconcomp_distribution_edges(bins=None)
```

Computes the distribution of the number of edges in each biconnected components by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is int (an integer), it specifies the number of equal-width bins. When *bins* is a list, it specifies bin ranges and should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is str (a string), it specifies the method to use for calculating the bin ranges. For example, 'stone','auto','doane','fd','rice','scott','sqrt','sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of biconnected components whose edge count is within the range of the bin.

biconcomp_distribution_nodes

```
biconcomp_distribution_nodes(bins=None)
```

Computes the distribution of the number of nodes in each biconnected components by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is int (an integer), it specifies the number of equal-width bins. When *bins* is a list, it specifies bin ranges and should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of

a1 but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is str (a string), it specifies the method to use for calculating the bin ranges. For example, 'stone','auto','doane','fd','rice','scott','sqrt','sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of biconnected components whose node count is within the range of the bin.

biconcomp_size_edges

```
biconcomp_size_edges()
```

Computes the number of edges within each biconnected component.

Returns

dict
Returns a dictionary in which the keys are biconnected component IDs and the values are the number of edges within each component.

biconcomp_size_nodes

```
biconcomp_size_nodes()
```

Computes the number of nodes within each biconnected component.

Returns

dict
Returns a dictionary in which the keys are biconnected component IDs and the values are the number of nodes within each component.

biconnected_component_count

```
biconnected_component_count()
```

Computes a graph's biconnected component count.

Returns

int
Returns the number of biconnected components.

biconnected_components

```
biconnected_components(order_by=None, *, data=True, ascending=False)
```

Computes the graph's biconnected components for an undirected graph.

Parameters

order_by : 'node' or 'edge' or None, optional
Specifies the parameter that is used to sort the components by the number of nodes or edges within each component. The default is None, which means that the components are not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Returns the result as a generator of biconnected component subgraphs. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

block_cut_tree

```
block_cut_tree(*, return_nodes_map=False)
```

Computes the block-cut tree of a graph.

Parameters

return_nodes_map : bool, optional
Specifies whether or not to also compute the block-cut tree node map. The default is False.

Returns

SAS graph object
Returns a graph that represents the block-cut tree. When *return_nodes_map = True*, the output graph has the *nodes_map* attribute, which contains a dictionary that maps its nodes to a list of nodes in the original graph. Note: If a block is empty (that is, the corresponding biconnected component contains only articulation points), that block does not appear as a key in the *nodes_map* dictionary.

centrality_betweenness

```
centrality_betweenness(edge_weight=None, sample_percent=100, *,
inplace=False, data=True)
```

Computes weighted or unweighted betweenness centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that unweighted betweenness centrality is computed.

sample_percent : int, optional
Specifies the percentage of source nodes to sample for the approximate betweenness calculation. This parameter value should be in the range (0, 100]. The default is 100.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the node and edge attributes *between_** . This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_closeness

```
centrality_closeness(edge_weight=None, nopath='diameter', *, inplace=False,
data=True)
```

Computes weighted or unweighted closeness centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted closeness centrality is computed.

nopath : {'diameter', 'harmonic', or 'zero'}, optional
Specifies a method to use for accounting for the shortest path distance between two nodes when a path does not exist. The valid values are as follows: - 'diameter' uses the graph diameter (plus one) as the shortest path distance between disconnected nodes. - 'harmonic' uses the harmonic formula for calculating closeness centrality. - 'zero' uses zero as the shortest path distance between disconnected nodes.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the closeness centrality node attributes *centr_close_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_degree

```
centrality_degree(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted degree centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted degree centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the degree centrality node attributes *centr_degree_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_eigenvector

```
centrality_eigenvector(edge_weight=None, maxiters=10000, *, inplace=False,
data=True)
```

Computes weighted or unweighted eigenvector centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted eigenvector centrality is computed.

maxiters : int, optional
Specifies the maximum number of iterations in order to limit the amount of computation time that is spent when convergence is slow. The default is 10,000.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the degree centrality node attributes *centr_eigen_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_hub_authority

```
centrality_hub_authority(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted hub and authority centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted hub and authority centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the hub and authority centrality node attributes *centr_hub_** and *centr_auth_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_influence

```
centrality_influence(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted influence centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted influence centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the first- and second-order influence centrality node attributes *centr_influence1_** and *centr_influence2_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_pagerank

```
centrality_pagerank(edge_weight=None, alpha=0.85, tolerance=1e-09, *,
inplace=False, data=True)
```

Computes weighted or unweighted PageRank centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted PageRank centrality is computed.

alpha : float, optional
Specifies the damping parameter for the PageRank centrality (or algorithm). The default is 0.85.

tolerance : float, optional
Specifies the tolerance parameter for the PageRank centrality (or algorithm). The default is 1E-9.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the PageRank centrality node attributes *centr_pagerank_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

concomp_distribution_edges

```
concomp_distribution_edges(bins=None, *, strongly=True)
```

Computes the distribution of the number of edges in each connected component by using *numpy.histogram*.

Parameters

bins : int or list or str, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is an integer (int), it represents the number of equal-width bins. When *bins* is a list, it specifies bin ranges and it should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is a string (str), it specifies the method to be used for calculating the bin ranges. For example, 'stone', 'auto', 'doane', 'fd', 'rice', 'scott', 'sqrt', 'sturges'. See https://

numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of components whose edge count within the range of the bin.

concomp_distribution_nodes

```
concomp_distribution_nodes(bins=None, *, strongly=True)
```

Computes the distribution of the number of nodes in each connected component by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is an integer (int), it represents the number of equal-width bins. When *bins* is a list, it specifies bin ranges and it should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is a string (str), it specifies the method to use for calculating the bin ranges. For example, 'stone', 'auto', 'doane', 'fd', 'rice', 'scott', 'sqrt', 'sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of components whose node count is within the range of the bin.

concomp_size_edges

```
concomp_size_edges(*, strongly=True)
```

Computes the number of edges within each connected component.

Parameters

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary in which the keys are connected component IDs and the values are the number of edges within each component.

concomp_size_nodes

```
concomp_size_nodes(*, strongly=True)
```

Computes the number of nodes within each connected component.

Parameters

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary in which the keys are connected component IDs and the values are the number of nodes within each component.

connected_component_count

```
connected_component_count()
```

Computes a graph's connected component count.

Returns

int
Returns the number of connected components.

connected_components

```
connected_components(order_by=None, *, strongly=True, data=True,
ascending=False)
```

Computes the connected components of a graph.

Parameters

order_by : 'node' or 'edge' or None, optional
Specifies whether to sort the components by the number of nodes or edges within each component. The default is None, which means that the components are not sorted.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one connected component. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

copy

```
copy(immutable=None, *, data=True)
```

Returns a copy of the input graph.

Parameters

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, meaning that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns an independent copy of the input.

core_decomposition

```
core_decomposition(maxtime=None, *, data=True)
```

Computes a core decomposition of the graph.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time for core decomposition to spend. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one k-core in increasing core number. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

cycle_count

```
cycle_count(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Counts the number of cycles in a graph.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Returns

int
Returns a count of cycles in the graph.

degree

```
degree(n=None, edge_weight=None)
```

Returns the degree of nodes.

Parameters

n : single node or container of nodes or None, optional
Specifies which nodes to include. The default is None, which means that the degree of all nodes is returned.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None.

Returns

int or dictionary
Returns the degree of specified node(s).

density

```
density()
```

Computes the density of the graph.

Returns

float
Returns the density of the graph.

diameter

```
diameter(edge_weight=None)
```

Computes a graph's diameter.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted diameter is computed.

Returns

int or float
Returns the diameter.

drop_edge_attributes

```
drop_edge_attributes(attributes_subset=None, edges_subset=None,
immutable=None, *, inplace=False)
```

Removes edge attributes if they exist.

Parameters

attributes_subset : str or list, optional
Specifies the edge attribute or a list of the edge attributes to remove; for example, 'weight' or ['weight','color']. By default, all edge attributes are removed.

edges_subset : list or iterable of edges, optional
Specifies the edges to remove attributes from; for example, [(1,2),(2,3)]. By default the attributes for all edges are removed.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by removing the specified edge attributes. If the *inplace* parameter value is False, a copy of the input graph with the modified edge attributes is returned.

drop_node_attributes

```
drop_node_attributes(attributes_subset=None, nodes_subset=None,
immutable=None, *, inplace=False)
```

Removes node attributes if they exist.

Parameters

attributes_subset : str or list, optional
Specifies the node attribute or list of node attributes to remove; for example, ['weight','color']. By default, all node attributes are removed.

nodes_subset : list or iterable of nodes, optional
Specifies the nodes to drop attributes from; for example, [1,2,3] or (1,2,3) or [1] or (1,). By default, the attributes for all nodes are removed.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by removing the specified node attributes. If the *inplace* parameter value is False, a copy of the input graph with the modified node attributes is returned.

enumerate_cycle_edges

```
enumerate_cycle_edges(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Enumerates the cycles in a graph and returns the edges of the cycles.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Yields

generator
Yields a generator of cycles, each of which is given as a list of the cycles' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_cycle_nodes

```
enumerate_cycle_nodes(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Enumerates the cycles in a graph and returns the nodes of the cycles.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Yields

generator
Yields a generator of cycles, each of which is given as a list of the cycles' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_cycles

```
enumerate_cycles(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None, *,
data=True)
```

Enumerates the cycles in a graph and returns the cycle graph objects.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one cycle of the same type as the input graph. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_path_edges

```
enumerate_path_edges(source=None, sink=None, minlength=1,
maxlength=None, edge_weight=None, node_weight=None, minedgewt=None,
maxedgewt=None, minnodewt=None, maxnodewt=None, maxtime=None)
```

Enumerates the paths in a graph and returns the edges of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

Yields

generator
Yields a generator of Path objects, each of which is given as a list of the paths' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_path_nodes

```
enumerate_path_nodes(source=None, sink=None, minlength=1,
maxlength=None, edge_weight=None, node_weight=None, minedgewt=None,
maxedgewt=None, minnodewt=None, maxnodewt=None, maxtime=None)
```

Enumerates the paths in a graph and returns the nodes of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

Yields

generator
Yields a generator of Path objects, each of which is given as a list of the paths' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_paths

```
enumerate_paths(source=None, sink=None, minlength=1, maxlength=None,
edge_weight=None, node_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, *, data=True)
```

Enumerates the paths in a graph and returns the Path graph objects.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one Path object. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_sequence_shortest_paths

```
enumerate_sequence_shortest_paths(sequence, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None, *, data=True)
```

Enumerates the shortest paths in a graph that visit the indicated sequence nodes in order.

Parameters

sequence : iterable of nodes
Specifies the order in which to visit the required nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Path objects; each object represents a subpath between a sequence of nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_path_edges

```
enumerate_shortest_path_edges(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None)
```

Enumerates the shortest paths in a graph and returns the edges of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

Yields

generator
Yields a generator of paths; each path is given as a list of the paths' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_path_nodes

```
enumerate_shortest_path_nodes(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None)
```

Enumerates the shortest paths in a graph and returns the nodes of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

Yields

generator
Yields a generator of paths; each path is given as a list of the paths' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_paths

```
enumerate_shortest_paths(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None, *, data=True)
```

Enumerates the shortest paths in a graph and returns the Path graph objects.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Path objects; each object represents one shortest path. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

get_meta_graph

```
get_meta_graph(node_type, edge_type=None, immutable=None)
```

Constructs a metagraph from the input graph.

Parameters

node_type : str
Specifies the node attribute to consider in constructing the metagraph.

edge_type : str, optional
Specifies the edge attribute to consider in constructing the metagraph. By default, all connections between nodes are considered the same.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

Returns

SAS graph object
Returns the metagraph, which has an edge attribute count and determines the number of unique connections between pairs of nodes.

get_nlargest_biconcomps

```
get_nlargest_biconcomps(n, order_by='node', *, data=True, ascending=False)
```

Returns a list of up to *n* largest biconnected components.

Parameters

n : int
Specifies the number of graphs to return.

order_by : 'node' or 'edge' or None
Specifies whether to sort the biconnected components by the number of nodes or edges within those components. The default is None, which means that the biconnected components are not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. When the value is True, the smallest biconnected components are returned instead. The default is False.

Returns

list
Returns a list of n largest or smallest biconnected components of a graph.

get_nlargest_concomps

```
get_nlargest_concomps(n, order_by='node', *, strongly=True, data=True,
ascending=False)
```

Returns a list of up to *n* largest connected (with highest number of nodes) components of a graph.

Parameters

n : int
Specifies the number of graphs to return.

order_by : 'node' or 'edge'
Specifies whether to sort the component by number of nodes or edges within that component. The default is None, which means that the result is not sorted.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Returns

list
Returns a list of the n largest connected components of a graph.

has_edge_attribute

```
has_edge_attribute(name, *, all_edges=False)
```

Checks the edges of the input graph for the specified edge attribute.

Parameters

name : str
Specifies the name of the edge attribute.

all_edges : bool, optional
Specifies whether or not to check for the existence of the edge attribute in all or any of the edges. The default is False.

Returns

bool
If the all_edges parameter value is True, the value True is returned if that edge attribute exists in all edges; otherwise the value False is returned. If the all_edges parameter value is False, the value True is returned if at least one edge in the graph has that specified edge attribute.

has_node_attribute

```
has_node_attribute(name, *, all_nodes=False)
```

Checks the nodes of the input graph for the specified node attribute.

Parameters

name : str
Specifies the name of the node attribute.

all_nodes : bool, optional
Specifies whether or not to check for the existence of the node attribute in all or any of the nodes. The default is False.

Returns

bool
If the all_nodes parameter value is True, the value True is returned if that node attribute exists in all node; otherwise the value False is returned. If the all_nodes parameter value is False, the value True is returned if at least one node in the graph has that specified node attribute.

intersection

```
intersection(rhs, nodes_attrs_agg=None, edges_attrs_agg=None)
```

Returns the intersection of input graphs by finding the intersections of their nodes and edges.

The node attributes and edge attributes are aggregated using the rules that are defined by the *nodes_attrs_agg* and *edges_attrs_agg* parameters.

Parameters

rhs : SAS graph object
Specifies the other graph, of the same type as *self*, with which to intersect.

nodes_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate node attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps node attributes to functions. The default is None.

edges_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate edge attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps edge attributes to functions. The default is None.

Returns

SAS graph object
Returns a graph that is constructed by the intersection of the two graphs.

is_biconnected

```
is_biconnected()
```

Specifies whether or not the input graph is biconnected.

Returns

bool
Indicates whether the input graph is biconnected (True) or not (False).

is_bipartite

```
is_bipartite()
```

Determines whether the input graph is bipartite.

Returns

bool
Indicates whether the input graph is bipartite.

is_connected

```
is_connected()
```

Specifies whether or not the input graph is connected.

Returns

bool
Indicates whether the input graph is connected (True) or not (False).

is_cycle

```
is_cycle()
```

Determines whether the input graph is a simple cycle.

Returns

bool
Indicates whether the input graph is a simple cycle (True) or not (False).

is_dag

```
is_dag()
```

Determines whether the input graph is a directed acyclic graph (DAG).

Returns

bool
Indicates whether the input graph is a directed acyclic graph (DAG).

is_directed

```
is_directed()
```

Specifies whether or not the graph is directed.

Returns

bool
Indicates whether or not the graph is directed.

is_hamiltonian

```
is_hamiltonian(maxtime=None)
```

Determines whether the graph is Hamiltonian (has a Hamiltonian cycle) or not.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time to determine whether or not the graph is Hamiltonian. The default is None.

Returns

bool or None
Indicates whether or not the input graph is determined to be Hamiltonian. Returns None if no determination is made.

is_immutable

```
is_immutable()
```

Specifies whether or not the graph is immutable.

Returns

bool
Indicates whether or not the graph is immutable.

is_multigraph

```
is_multigraph()
```

Specifies whether or not the graph allows multiedges.

Returns

bool
Indicates whether or not the graph allows multiedges.

is_path

```
is_path()
```

Determines whether the input graph is a simple path.

Returns

bool
Indicates whether the input graph is a simple path (True) or not (False).

is_strongly_connected

```
is_strongly_connected()
```

Specifies whether the input graph is strongly connected.

Returns

bool
Indicates whether the input graph is strongly connected (True) or not (False).

is_weakly_connected

```
is_weakly_connected()
```

Specifies whether the input graph is weakly connected.

Returns

bool
Indicates whether the input graph is weakly connected (True) or not (False).

label_propagation

```
label_propagation(edge_weight=None, maxiters=100, tolerance=0.05, *,
data=True)
```

Computes the communities of the graph by using the label propagation algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of label changes for all nodes in the graph is less than the value. The default is 0.05.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one community. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

label_propagation_nodes

```
label_propagation_nodes(edge_weight=None, maxiters=100, tolerance=0.05, *,
data=True, inplace=False)
```

Computes community assignments for all nodes in the graph by using the label propagation algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of label changes for all nodes in the graph is less than the value. The default is 0.05.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

inplace : bool, optional
Specifies whether to add the community values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains the node attribute *community*, which denotes the community assignment for each node. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

leaf_node_count

```
leaf_node_count()
```

Computes a graph's leaf node count.

Returns

int
Returns the number of leaf nodes.

local_transitivity

```
local_transitivity(*, inplace=False)
```

Computes the local transitivity (clustering coefficient) and stores the results as node attributes on the graph.

Parameters

inplace : bool, optional
Specifies whether to add the transitivity values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains node attribute 'transitivity'. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

louvain

```
louvain(*, edge_weight=None, resolution=1, tolerance=0.001, maxiters=100,
data=True)
```

Computes the communities of the graph by using the Louvain algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

resolution : float, optional
Specifies the factor in the modularity equation that affects the expected number of links between two nodes. A higher value produces more communities. The default is 1.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of modularity gain between two consecutive iterations is less than the value. The default is 0.001.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one community. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

louvain_nodes

```
louvain_nodes(*, edge_weight=None, resolution=1, tolerance=0.001,
maxiters=100, data=True, inplace=False)
```

Computes community assignments for all nodes in the graph by using the Louvain algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

resolution : float, optional
Specifies the factor in the modularity equation that affects the expected number of links between two nodes. A higher value produces more communities. The default is 1.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of modularity gain between two consecutive iterations is less than the value. The default is 0.001.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

inplace : bool, optional
Specifies whether to add the community values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains the node attribute *community*, which denotes the community assignment for each node. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

maxflow

```
maxflow(source, sink, capacity_attr='upper', *, inplace=False, data=True)
```

Calculates the maximum flow from a source node to a sink node in a graph.

Parameters

source : number or str
Specifies the source node for the maximum flow.

sink : number or str
Specifies the sink node for the maximum flow.

capacity_attr : str, optional
Specifies the name of the edge attribute that contains the capacity of the edge. The default is 'upper'.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute *flow*, which contains the flow values. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, the returned graph contains only the flow edges.

maximal_cliques

```
maximal_cliques(maxcliques='ALL', node_weight=None, edge_weight=None,
minsize=1, maxsize=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, order_by=None, *,
data=True, ascending=False)
```

Enumerates all maximal cliques of the graph.

Parameters

maxcliques : int or 'ALL', optional
Specifies the maximum number of enumerated cliques to return. The default is 'ALL'.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

minsize : int, optional
Specifies the minimum number of nodes in a clique. The default is 1.

maxsize : int or None, optional
Specifies the maximum number of nodes in a clique. The default is None.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a clique. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a clique. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a clique. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a clique. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds for enumeration to spend. The default is None.

order_by : 'node' or None, optional
Specifies whether to sort the cliques by the number of nodes within each clique. The default is None, which means that the result is not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one maximal clique. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

min_cut

```
min_cut(edge_weight=None, *, inplace=False, data=True)
```

Calculates a minimum cut of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted minimum cut is calculated.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes

of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute 'mincut', which indicates the edges to include in the minimum cut set, and the node attribute 'mincut_partition', which indicates the node partition of the calculated minimum cut. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, a modified copy of the graph is returned.

min_s_t_cut

```
min_s_t_cut(source, sink, edge_weight=None, *, inplace=False, data=True)
```

Calculates a minimum s-t cut of a graph.

Parameters

source : number or str
Specifies the source node (s) for the minimum s-t cut.

sink : number or str
Specifies the sink node (t) for the minimum s-t cut.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted minimum s-t cut is calculated.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute *s_t_cut*, which indicates the edges to include in the minimum s-t cut set; and the node attribute *s_t_cut_partition*, which indicates the node partition of the calculated minimum s-t cut. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is

modified in place and None is returned. When the *inplace* parameter value is False, a modified copy of the graph is returned.

mincostflow

```
mincostflow(demand_lower_attr='lower', demand_upper_attr='upper',
capacity_lower_attr='lower', capacity_upper_attr='upper', cost_attr='weight',
maxtime=None, *, inplace=False, data=True)
```

Calculates the minimum-cost network flow for the graph.

Parameters

demand_lower_attr : str, optional
Specifies the name of the node attribute that contains the lower bound of the node supply. The default is 'lower'.

demand_upper_attr : str, optional
Specifies the name of the node attribute that contains the upper bound of the node supply. The default is 'upper'.

capacity_lower_attr : str, optional
Specifies the name of the edge attribute that contains the capacity lower bound of the edge. The default is 'lower'.

capacity_upper_attr : str, optional
Specifies the name of the edge attribute that contains the capacity upper bound of the edge. The default is 'upper'.

cost_attr : str, optional
Specifies the name of the edge attribute that contains the edge cost. The default is 'weight'.

maxtime : float or None, optional
Specifies the maximum amount of time to allow for computing the minimum-cost network flow. The default is None.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attributes *flow* and *reduced_cost*, which contain the flow and reduced cost values, respectively; and the node attribute *dual*, which contains the optimal dual value. This graph has the

solution_summary attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned.

minimum_spanning_tree

```
minimum_spanning_tree(edge_weight=None, source=None, *, inplace=False,
data=True)
```

Calculates a minimum spanning tree (MST) or forest of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

source : number or str or None
Specifies the source node for a directed MST. The default is None.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute 'mst', which indicates the edges to include in the spanning tree or forest. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, the returned graph contains only the spanning tree or forest edges.

minimum_weight_full_matching

```
minimum_weight_full_matching(edge_weight=None, *, data=True)
```

Returns a minimum-weight full matching of the bipartite graph *self*.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns a directed graph whose edges identify the assignment mapping in the minimum-weight full matching solution. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

neighbors

```
neighbors(n)
```

Returns an iterator over all neighbors of node n.

Parameters

n : node
Specifies a node in the graph.

Returns

iterator
Returns an iterator over all neighbors of node n.

node_clique_numbers

```
node_clique_numbers(*, inplace=False, data=True)
```

Computes node clique numbers and stores the results as attributes on the graph.

Parameters

inplace : bool, optional
Specifies whether to add the clique number values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the value of the *inplace* parameter is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph or modifies the existing graph to contain the node attribute *clique_number*, which includes the clique number of each node. The graph g has the attribute g.solution_summary, which contains information about the results of the algorithm.

number_of_edges

```
number_of_edges(u=None, v=None)
```

Returns the number of edges in the graph.

When you specify *u* and *v*, the number of edges between *u* and *v* is returned. When you specify *u* but *v* is None, the number of edges that connect to *u* is returned.

Parameters

u : numeric or str or None, optional
Specifies the *from* node. The default is None.

v : numeric or str or None, optional
Specifies the *to* node. The default is None.

Returns

int
Returns the number of edges in the graph. When you specify *u* and *v*, the number of edges between *u* and *v* is returned. When you specify *u* but *v* is None, the number of edges that connect to *u* is returned.

number_of_nodes

```
number_of_nodes()
```

Returns the number of nodes in the graph.

Returns

int
Returns the number of nodes in the graph.

projected_graph

```
projected_graph(nodes, neighbors=None, directed_method=None,
_measure=None, *, multigraph=False, data=True)
```

Computes a projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in self other than *nodes*.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used. When the *directed_method* parameter is specified for an undirected graph, a ValueError is raised.

multigraph : bool, optional
Specifies whether or not to return a multigraph. When the value is True, the algorithm returns a multigraph in which the multiple edges represent multiple shared neighbors. The edge attribute 'neighbor' stores the label of the shared neighbor. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

projected_graph_adamic_adar

```
projected_graph_adamic_adar(nodes, neighbors=None, directed_method=None,
*, data=True)
```

Computes an Adamic-Adar projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than *nodes*.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'adamic_adar' edge attribute.

projected_graph_common_neighbors

```
projected_graph_common_neighbors(nodes, neighbors=None,
directed_method=None, *, data=True)
```

Computes a common neighbors projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'common_neighbors' edge attribute.

projected_graph_cosine

```
projected_graph_cosine(nodes, edge_weight=None, neighbors=None,
directed_method=None, *, data=True)
```

Computes a cosine projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'cosine' edge attribute.

projected_graph_dice

```
projected_graph_dice(nodes, neighbors=None, directed_method=None, *,
data=True)
```

Computes a Sørensen-Dice projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the subset of nodes that are specified by the *neighbors* parameter. The strength of association in the calculated projection is represented by the 'dice' edge attribute.

projected_graph_jaccard

```
projected_graph_jaccard(nodes, neighbors=None, directed_method=None, *,
data=True)
```

Computes a Jaccard projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'jaccard' edge attribute.

query

```
query(q, expr=None, _node_attr_to_ignore=None, _edge_attr_to_ignore=None, *,
induced=False, break_symmetry=False, _generate_output=True,
_orbits_only=False, **kwargs)
```

Queries for the subgraph isomorphisms of q within g.

Parameters

q : graph
Specifies the query graph.

expr : _Expression or None, optional
Specifies additional logic to filter the query results. The default is None.

induced : bool, optional
Specifies whether or not to return only node-induced matches. The default is False.

break_symmetry : bool, optional
Specifies whether or not to break symmetry and eliminate automorphic relationships between matches. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one isomorphic mapping from *q* to *self*. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

reach_graph

```
reach_graph(source, max_reach=1, *, data=True)
```

Computes the reach graph for the source or set of sources.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to compute the reach network.

max_reach : int, optional
Specifies the number of steps (or hops) to traverse from the source nodes. The default is 1.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns a graph that represents the reach network. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

reach_graph_per_source

```
reach_graph_per_source(source, max_reach=1, *, data=True)
```

Computes a separate reach graph for each given source node.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to compute the reach network.

max_reach : int, optional
Specifies the number of steps (or hops) to traverse from the source nodes. The default is 1.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one reach network. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

set_edge_attributes

```
set_edge_attributes(values, name=None, immutable=None, *, inplace=False)
```

Sets edge attributes by using a given value or dictionary of values.

Parameters

values : scalar or dict-like or Pandas DataFrame
Specifies what the edge attribute should be set to. When this parameter is set to a Pandas DataFrame, it is expected to have from, to, and edge_key (only for multigraphs) columns, and the other columns are treated as edge attributes to add. When *values* is a dictionary, the keys are edges and the values are either edge attribute values or dictionaries of attribute name-value pairs. For multigraphs, the edge tuples must be of the form (u, v, key). For simple graphs, the keys must be tuples of the form (u, v). When this parameter is not set to a dictionary, it is treated as a single attribute value that is then applied to every edge in *self*. The attribute name is specified by the *name* parameter.

name : str or None, optional
Specifies the name of the edge attribute to set if the *values* parameter is set to a scalar. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is False, a copy of the input graph with the new edge attributes is returned. If the *inplace* parameter value is True, the graph is modified by setting the specified edge attributes.

set_node_attributes

```
set_node_attributes(values, name=None, immutable=None, *, inplace=False)
```

Sets node attributes from a given value or dictionary of values.

Parameters

values : scalar or dict-like or Pandas DataFrame
Specifies what the node attribute should be set to. When *values* is not a dictionary, it is treated as a single attribute value that is then applied to every node in *self*. This means that if you provide a mutable object, such as a list, updates to that object are reflected in the node attribute for every node. The attribute name is specified by the *name* parameter. When *values* is a dictionary, the keys are nodes and the values are either node attribute values or dictionaries of attribute name-value pairs.

name : str or None, optional
Specifies the name of the node attribute to set if the *values* parameter is set to a scalar. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not the graph is modified in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by setting the specified node attributes. If the *inplace* parameter value is False, a copy of the input graph with the new node attributes is returned.

shortest_path_distances

```
shortest_path_distances(source=None, sink=None, edge_weight=None,
minedgewt=None, maxedgewt=None)
```

Calculates the shortest path distances in a graph.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) and restricts enumeration to the shortest paths that contain the specified source node(s). The default is None.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) and restricts enumeration to the shortest paths that contain the specified sink node(s). The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

Returns

dict
Returns a dictionary where each key is a (source, sink) tuple and its value is the corresponding shortest source-sink path distance.

similarity_adamic_adar

```
similarity_adamic_adar(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Adamic-Adar node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_common_neighbors

```
similarity_common_neighbors(source, sink=None, top_k=None, bottom_k=None,
*, exclude_source=False, exclude_existing_neighbors=False)
```

Computes common neighbors node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_cosine

```
similarity_cosine(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes cosine node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_dice

```
similarity_dice(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Sørensen-Dice node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_graph

```
similarity_graph(measure, min_score=2.220446049250313e-16, max_score=inf,
*, exclude_source=False, exclude_existing_neighbors=False, data=True,
**kwargs)
```

Computes a node similarity graph.

Parameters

measure : str, one of {'adamic_adar', 'common_neighbors', 'cosine', 'dice', 'jaccard'}
Specifies the metric to be used to compute the node similarity.

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the edge attribute that corresponds to the measure name. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_adamic_adar

```
similarity_graph_adamic_adar(min_score=2.220446049250313e-16,
max_score=inf, *, exclude_source=False, exclude_existing_neighbors=False,
data=True)
```

Computes an Adamic-Adar node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'adamic_adar' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_common_neighbors

```
similarity_graph_common_neighbors(min_score=2.220446049250313e-16,
max_score=inf, *, exclude_source=False, exclude_existing_neighbors=False,
data=True)
```

Computes a common neighbors node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'common_neighbors' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_cosine

```
similarity_graph_cosine(edge_weight=None,
min_score=2.220446049250313e-16, max_score=inf, *, exclude_source=False,
exclude_existing_neighbors=False, data=True)
```

Computes a cosine node similarity graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'cosine' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_dice

```
similarity_graph_dice(min_score=2.220446049250313e-16, max_score=inf, *,
exclude_source=False, exclude_existing_neighbors=False, data=True)
```

Computes a Sørensen-Dice node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the *dice* edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_jaccard

```
similarity_graph_jaccard(min_score=2.220446049250313e-16, max_score=inf, *,
exclude_source=False, exclude_existing_neighbors=False, data=True)
```

Computes a Jaccard node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'jaccard' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_jaccard

```
similarity_jaccard(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Jaccard node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

singleton_node_count

```
singleton_node_count()
```

Computes a graph's singleton node count.

Returns

int
Returns the number of singleton nodes.

to_directed

```
to_directed()
```

Returns a directed copy of the input graph.

Returns

SAS graph object
Returns a directed copy of the input graph.

to_immutable

```
to_immutable()
```

Creates an immutable graph from the input graph.

Returns

SAS graph object
Returns an immutable SAS graph object.

to_mutable

```
to_mutable()
```

Creates a mutable graph from the input graph.

Returns

SAS graph object
Returns a mutable SAS graph object.

to_pandas_edges

```
to_pandas_edges(*, data=True, _for_conversion_to_immutable=False)
```

Returns the graph edges as a Pandas DataFrame.

Parameters

data : bool or list, optional
Specifies whether or not to retain attributes. The default is True. When the value is True (or when a list is given), the Pandas DataFrame includes all the edge attributes (or all the listed edge attributes).

Returns

dataframe
Returns a Pandas DataFrame that includes the graph edges.

to_pandas_nodes

```
to_pandas_nodes(*, data=True, _for_conversion_to_immutable=False)
```

Returns the graph nodes as a Pandas DataFrame.

Parameters

data : bool or list, optional
Specifies whether or not to retain attributes. The default is True. When the value is True (or when a list is given), the Pandas DataFrame includes all the node attributes (or all the listed node attributes).

Returns

dataframe
Returns a Pandas DataFrame that includes the graph nodes.

to_undirected

```
to_undirected()
```

Returns a directed copy of the input graph.

Returns

SAS graph object
Returns a directed copy of the input graph.

topological_ordering

```
topological_ordering()
```

Finds a topological ordering of the graph's nodes.

Yields

generator
Yields a generator of nodes in topological order. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

transitive_closure

```
transitive_closure()
```

Calculates the transitive closure of a graph.

Returns

SAS graph object
Returns the transitive closure of the input graph that is returned as a SAS MultiGraph or MultiDiGraph object. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

triangle_count

```
triangle_count()
```

Computes a graph's triangle count.

Returns

int
Returns the number of triangles.

tsp_tour

```
tsp_tour(edge_weight=None, maxtime=None, minobjective=None, *,
inplace=False, use_milp=True, data=True, **options)
```

Calculates a traveling salesman problem (TSP) tour of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted TSP is calculated.

maxtime : float or None, optional
Specifies the maximum amount of time for solving a TSP. The default is None.

minobjective : float, optional
Specifies a minimum value such that when a solution is found whose objective is less than or equal to this value, the algorithm stops.

inplace : bool, optional
Specifies whether to add the TSP order values as attributes to the input graph directly or to create a copy of the graph. The default is False.

use_milp : bool, optional
Specifies whether or not to use a mixed integer linear programming (MILP) solver to solve a TSP. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is False, the returned graph is a Cycle that contains the calculated TSP tour edges. When the *inplace* parameter value is True, the graph is modified in place and None is returned. The modified graph contains the edge attributes *tsp* and *tsp_order* and the node attribute *tsp_order*.

union

```
union(rhs, nodes_attrs_agg=None, edges_attrs_agg=None)
```

Returns the union of input graphs by finding the unions of their nodes and edges.

The node attributes and edge attributes are aggregated using the rules that are defined by the *nodes_attrs_agg* and *edges_attrs_agg* parameters.

Parameters

rhs : SAS graph object
Specifies the other graph, of the same type as *self*, with which to find the union.

nodes_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate node attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps node attributes to functions. The default is None.

edges_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate edge attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps edge attributes to functions. The default is None.

Returns

SAS graph object
Returns a graph that is constructed by the union of the two graphs.

vrp

```
vrp(capacity, depot, node_demand='demand', edge_cost='weight', minroutes=1,
maxroutes=None, maxtime=None, minobjective=None, *, use_milp=True,
data=True, **options)
```

Solves the vehicle routing problem.

Parameters

capacity : float
Specifies the capacity of each vehicle.

depot : node
Specifies the depot node for the vehicle routing problem.

node_demand : str or dict or iterable
Specifies the quantity of goods to deliver to or pick up from a customer. The default is 'demand'.

edge_cost : str or dict or iterable
Specifies the cost of traversing each edge. The default is 'weight'.

minroutes : int, optional
Specifies the minimum number of routes that are allowed to service demand.

maxroutes : int, optional
Specifies the maximum number of routes that are allowed to service demand.

maxtime : float or None, optional
Specifies the maximum amount of time for solving the vehicle routing problem. The default is None.

minobjective : float, optional
Specifies a minimum value such that when a solution is found whose objective is less than or equal to this value, the algorithm stops.

use_milp : bool, optional
Specifies whether or not to use a mixed integer linear programming (MILP) solver to solve the vehicle routing problem. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Cycle objects; each cycle represents one route. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

vrptw

```
vrptw(capacity, depot, node_demand='demand', edge_cost='weight',
travel_time='weight', timewindow_lower=None, timewindow_upper=None,
service_time=None, minroutes=1, maxroutes=None, maxtime=None, *,
data=True)
```

Solves the vehicle routing problem with time windows.

Parameters

capacity : float
Specifies the capacity of each vehicle.

depot : node
Specifies the depot node for the vehicle routing problem.

node_demand : str or dict or iterable
Specifies the quantity of goods to deliver to or pick up from a customer. The default is 'demand'.

edge_cost : str or dict or iterable
Specifies the cost of traversing each edge.

travel_time : str or dict or iterable
Specifies attributes or values that define the time that it takes to traverse each edge. The default is 'weight'.

timewindow_lower : str or dict or iterable or None, optional
Specifies attributes or values that define the lower time limits within which a delivery vehicle can arrive at the customer nodes. The default is None.

timewindow_upper : str or dict or iterable or None, optional
Specifies attributes or values that define the upper time limits within which a delivery vehicle can arrive at the customer nodes. The default is None.

service_time : str or dict or iterable or None, optional
Specifies attributes or values that define the service time of the nodes. The default is None.

minroutes : int, optional
Specifies the minimum number of routes that are allowed to service demand.

maxroutes : int, optional
Specifies the maximum number of routes that are allowed to service demand.

maxtime : float or None, optional
Specifies the maximum amount of time for solving the vehicle routing problem. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Cycle objects; each cycle represents one route. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

24

Output Graph Classes

Classes

Class Name	Description
Cycle	Class for result graphs that are cycles.
Path	Class for result graphs that are paths.

Cycle

Class for result graphs that are cycles.

API: Cycle

class sasviya.network.classes.cycle.Cycle(input_graph=None, data=None, edge_weight=None, first_edge=None, *args, **kwargs)

Parameters

input_graph : SAS graph object or None, optional
Specifies the underlying graph for a Cycle graph. The default is None.

data : bool or None, optional
Specifies whether or not to copy data from the input graph. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute to use for calculating cycle length. The default is None, which means that unit edge weights are used.

first_edge : edge or None, optional
Specifies the starting place for iterating over the cycle's edges. The default is None.

Attributes

edges
Returns the edges of the graph.

nodes
Returns the nodes of the graph.

Methods

Method Name	Description
add_core_number_attr	Computes the core numbers and stores the results as node attributes on the graph.
add_edge	Adds a single edge to the graph.
add_edges_from	Adds specified edges to the graph.
add_node	Adds a node to the graph.
add_nodes_from	Adds the specified nodes to the graph.
add_topological_order_attr	Computes a topological ordering and stores the results as node attributes of the graph.
add_weighted_edges_from	Adds weighted edges to the graph.
adjacency	Returns an iterator over (node, adjacency dict) for all nodes.

Method Name	Description
allows_multiedges	Specifies whether or not the graph allows multiedges.
allows_selfloops	Specifies whether or not the graph allows self-loops.
apply	Applies a function to all node or edge attributes and updates them.
approximate_diameter	Computes an approximation of the graph's diameter.
articulation_point_count	Computes a graph's articulation point count.
articulation_points	Finds the articulation points (cut points) of a graph.
assortativity_degree	Computes a graph's degree assortativity.
assortativity_nominal	Computes a graph's nominal assortativity.
assortativity_numeric	Computes a graph's numeric assortativity.
average_shortest_path	Computes a graph's average shortest path.
biconcomp_distribution_edges	Computes the distribution of the number of edges in each biconnected components by using *numpy.histogram*.
biconcomp_distribution_nodes	Computes the distribution of the number of nodes in each biconnected components by using *numpy.histogram*.
biconcomp_size_edges	Computes the number of edges within each biconnected component.
biconcomp_size_nodes	Computes the number of nodes within each biconnected component.
biconnected_component_count	Computes a graph's biconnected component count.
biconnected_components	Computes the graph's biconnected components for an undirected graph.
block_cut_tree	Computes the block-cut tree of a graph.
centrality_betweenness	Computes weighted or unweighted betweenness centrality and stores the results as attributes on the graph.
centrality_closeness	Computes weighted or unweighted closeness centrality and stores the results as attributes on the graph.

Method Name	Description
centrality_degree	Computes weighted or unweighted degree centrality and stores the results as attributes on the graph.
centrality_eigenvector	Computes weighted or unweighted eigenvector centrality and stores the results as attributes on the graph.
centrality_hub_authority	Computes weighted or unweighted hub and authority centrality and stores the results as attributes on the graph.
centrality_influence	Computes weighted or unweighted influence centrality and stores the results as attributes on the graph.
centrality_pagerank	Computes weighted or unweighted PageRank centrality and stores the results as attributes on the graph.
clear	Removes all nodes and edges from the graph.
clear_edges	Removes all edges from the graph but retains nodes.
concomp_distribution_edges	Computes the distribution of the number of edges in each connected component by using *numpy.histogram*.
concomp_distribution_nodes	Computes the distribution of the number of nodes in each connected component by using *numpy.histogram*.
concomp_size_edges	Computes the number of edges within each connected component.
concomp_size_nodes	Computes the number of nodes within each connected component.
connected_component_count	Computes a graph's connected component count.
connected_components	Computes the connected components of a graph.
copy	Returns a copy of the input graph.
core_decomposition	Computes a core decomposition of the graph.
cycle_count	Counts the number of cycles in a graph.
cycle_length	Returns the length of the cycle.

Method Name	Description
degree	Returns the degree of nodes.
density	Computes the density of the graph.
diameter	Computes a graph's diameter.
drop_edge_attributes	Removes edge attributes if they exist.
drop_node_attributes	Removes node attributes if they exist.
enumerate_cycle_edges	Enumerates the cycles in a graph and returns the edges of the cycles.
enumerate_cycle_nodes	Enumerates the cycles in a graph and returns the nodes of the cycles.
enumerate_cycles	Enumerates the cycles in a graph and returns the cycle graph objects.
enumerate_path_edges	Enumerates the paths in a graph and returns the edges of the paths.
enumerate_path_nodes	Enumerates the paths in a graph and returns the nodes of the paths.
enumerate_paths	Enumerates the paths in a graph and returns the Path graph objects.
enumerate_sequence_shortest_paths	Enumerates the shortest paths in a graph that visit the indicated sequence nodes in order.
enumerate_shortest_path_edges	Enumerates the shortest paths in a graph and returns the edges of the paths.
enumerate_shortest_path_nodes	Enumerates the shortest paths in a graph and returns the nodes of the paths.
enumerate_shortest_paths	Enumerates the shortest paths in a graph and returns the Path graph objects.
get_meta_graph	Constructs a metagraph from the input graph.
get_nlargest_biconcomps	Returns a list of up to *n* largest biconnected components.
get_nlargest_concomps	Returns a list of up to *n* largest connected (with highest number of nodes) components of a graph.
has_edge_attribute	Checks the edges of the input graph for the specified edge attribute.

Method Name	Description
has_node_attribute	Checks the nodes of the input graph for the specified node attribute.
in_degree	Returns the in-degree of nodes.
intersection	Returns the intersection of input graphs by finding the intersections of their nodes and edges.
is_biconnected	Specifies whether or not the input graph is biconnected.
is_bipartite	Determines whether the input graph is bipartite.
is_connected	Specifies whether or not the input graph is connected.
is_cycle	Determines whether the input graph is a simple cycle.
is_dag	Determines whether the input graph is a directed acyclic graph (DAG).
is_directed	Specifies whether or not the graph is directed.
is_hamiltonian	Determines whether the graph is Hamiltonian (has a Hamiltonian cycle) or not.
is_immutable	Specifies whether or not the graph is immutable.
is_multigraph	Specifies whether or not the graph allows multiedges.
is_path	Determines whether the input graph is a simple path.
is_strongly_connected	Specifies whether the input graph is strongly connected.
is_weakly_connected	Specifies whether the input graph is weakly connected.
label_propagation	Computes the communities of the graph by using the label propagation algorithm.
label_propagation_nodes	Computes community assignments for all nodes in the graph by using the label propagation algorithm.
leaf_node_count	Computes a graph's leaf node count.
local_transitivity	Computes the local transitivity (clustering coefficient) and stores the results as node attributes on the graph.

Method Name	Description
louvain	Computes the communities of the graph by using the Louvain algorithm.
louvain_nodes	Computes community assignments for all nodes in the graph by using the Louvain algorithm.
maxflow	Calculates the maximum flow from a source node to a sink node in a graph.
maximal_cliques	Enumerates all maximal cliques of the graph.
min_cut	Calculates a minimum cut of a graph.
min_s_t_cut	Calculates a minimum s-t cut of a graph.
mincostflow	Calculates the minimum-cost network flow for the graph.
minimum_spanning_tree	Calculates a minimum spanning tree (MST) or forest of a graph.
minimum_weight_full_matching	Returns a minimum-weight full matching of the bipartite graph *self*.
neighbors	Returns an iterator over all neighbors of node n.
node_clique_numbers	Computes node clique numbers and stores the results as attributes on the graph.
number_of_edges	Returns the number of edges in the graph.
number_of_nodes	Returns the number of nodes in the graph.
out_degree	Returns the out-degree of nodes.
predecessors	Returns an iterator over all predecessor nodes of n.
projected_graph	Computes a projected graph.
projected_graph_adamic_adar	Computes an Adamic-Adar projected graph.
projected_graph_common_neighbors	Computes a common neighbors projected graph.
projected_graph_cosine	Computes a cosine projected graph.
projected_graph_dice	Computes a Sørensen-Dice projected graph.
projected_graph_jaccard	Computes a Jaccard projected graph.

Method Name	Description
query	Queries for the subgraph isomorphisms of q within g.
reach_graph	Computes the reach graph for the source or set of sources.
reach_graph_per_source	Computes a separate reach graph for each given source node.
remove_edge	Removes a single edge from the graph.
remove_edges_from	Removes specified edges from the graph.
remove_isolated_nodes	Removes isolated nodes from a graph.
remove_node	Removes a single node from the graph.
remove_nodes_from	Removes specified nodes from the graph.
reverse	Returns a copy of a directed graph with reversed edges.
set_edge_attributes	Sets edge attributes by using a given value or dictionary of values.
set_node_attributes	Sets node attributes from a given value or dictionary of values.
shortest_path_distances	Calculates the shortest path distances in a graph.
similarity_adamic_adar	Computes Adamic-Adar node similarity.
similarity_common_neighbors	Computes common neighbors node similarity.
similarity_cosine	Computes cosine node similarity.
similarity_dice	Computes Sørensen-Dice node similarity.
similarity_graph	Computes a node similarity graph.
similarity_graph_adamic_adar	Computes an Adamic-Adar node similarity graph.
similarity_graph_common_neighbors	Computes a common neighbors node similarity graph.
similarity_graph_cosine	Computes a cosine node similarity graph.
similarity_graph_dice	Computes a Sørensen-Dice node similarity graph.
similarity_graph_jaccard	Computes a Jaccard node similarity graph.

Method Name	Description
similarity_jaccard	Computes Jaccard node similarity.
singleton_node_count	Computes a graph's singleton node count.
successors	Returns an iterator over all successor nodes of n.
to_directed	Returns a directed copy of the input graph.
to_immutable	Creates an immutable graph from the input graph.
to_mutable	Creates a mutable graph from the input graph.
to_pandas_edges	Returns the graph edges as a Pandas DataFrame.
to_pandas_nodes	Returns the graph nodes as a Pandas DataFrame.
to_undirected	Returns a directed copy of the input graph.
topological_ordering	Finds a topological ordering of the graph's nodes.
transitive_closure	Calculates the transitive closure of a graph.
triangle_count	Computes a graph's triangle count.
tsp_tour	Calculates a traveling salesman problem (TSP) tour of a graph.
union	Returns the union of input graphs by finding the unions of their nodes and edges.
vrp	Solves the vehicle routing problem.
vrptw	Solves the vehicle routing problem with time windows.

add_core_number_attr

```
add_core_number_attr(maxtime=None, *, inplace=False, data=True)
```

Computes the core numbers and stores the results as node attributes on the graph.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time for core decomposition to spend. The default is None.

inplace : bool, optional
Specifies whether to add the core numbers as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph with core numbers that are stored as a node attribute named *core*. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

add_edge

```
add_edge(*args, **kwargs)
```

Adds a single edge to the graph.

Parameters

u : numeric or str
Specifies the *from* node.

v : numeric or str
Specifies the *to* node.

error_on_selfloop : bool or None, optional
Specifies whether or not to raise a ValueError if a self-loop is detected; that is, if (u==v). The default is None, which means that the input edge is silently ignored.

Examples

```
G.add_edge(1,2,weight=3,type='parent')
```

add_edges_from

```
add_edges_from(*args, **kwargs)
```

Adds specified edges to the graph.

Parameters

edges : an iterable container of edges
Specifies the edges to add to the graph.

error_on_selfloop : bool or None, optional
Specifies whether or not to raise a ValueError if a self-loop is detected; that is, if (u==v). The default is None, which means that the input self-loop is silently ignored.

Examples

```
G.add_edges_from([("A","B",{"weight":1,"color":"blue"}),
                      ("C","D",{"weight":2,"color":"red"})])
```

```
G.add_edges_from([['X', 'Y'],["Y","Z"], weight=1,color= "blue")
```

add_node

```
add_node(*args, **kwargs)
```

Adds a node to the graph.

Parameters

node : str or number or tuple
Specifies the ID of the node to add to the graph. If the value is a tuple, the first element should contain the node ID, and the second element is a dictionary that represents the node's attributes.

add_nodes_from

```
add_nodes_from(*args, **kwargs)
```

Adds the specified nodes to the graph.

Parameters

nodes : an iterable container of nodes
Specifies the nodes to add to the graph.

Examples

```
G.add_nodes_from([('X', {"weight":11,"color": "blue"}), ("Y", {"color": "blue"})])
G.add_nodes_from(('X', 'Y'), weight=1,color= "blue")
G.add_nodes_from(['X', 'Y'])
```

add_topological_order_attr

```
add_topological_order_attr(*, inplace=False, data=True)
```

Computes a topological ordering and stores the results as node attributes of the graph.

Parameters

inplace : bool, optional
Specifies whether or not to add the topological order values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a directed graph whose topological order is stored as a node attribute named *top_order*. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

add_weighted_edges_from

```
add_weighted_edges_from(*args, **kwargs)
```

Adds weighted edges to the graph.

Parameters

edges : an iterable container of edges
Specifies the weighted edges to add to the graph.

weight : str
Specifies the attribute name for the weighted edges to add. The default is 'weight'.

adjacency

```
adjacency()
```

Returns an iterator over (node, adjacency dict) for all nodes.

For directed graphs, only outgoing adjacencies are returned.

Returns

iterator
Returns an iterator over (node, adjacency dictionary) for all nodes in the graph.

allows_multiedges

```
allows_multiedges()
```

Specifies whether or not the graph allows multiedges.

Returns

bool
Indicates whether or not the graph allows multiedges.

allows_selfloops

```
allows_selfloops()
```

Specifies whether or not the graph allows self-loops.

Returns

bool
Indicates whether or not the graph allows self-loops.

apply

apply(node_attr_handler=None, edge_attr_handler=None, immutable=None, *, inplace=False)

Applies a function to all node or edge attributes and updates them.

#use apply to fillna >>> sasnet.apply(G, edge_attr_handler=lambda data:{'w':0 if 'w' not in data else data['w']},inplace=True) >>> assert G.edges[(1,2)]['w'] == 0 >>> print(G.nodes(data=True)) #use apply to negate an attribute >>> sasnet.apply(G, node_attr_handler=lambda data:{'x': -data['x']},inplace=True) >>> assert G.nodes[1]['x']== -10

Parameters

node_attr_handler : function or None, optional
Specifies the function to apply to node attributes. The default is None.

edge_attr_handler : function or None, optional
Specifies the function to apply to edge attributes. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the input graph. The default is False.

Returns

SAS graph object or None.
Returns the resulting graph after the function is applied, or returns nothing if the *inplace* parameter value is True.

Examples

```
import sasviya.network as sasnet
G = sasnet.Graph([[1,2,{}],[1,3,{'w':1}],[1,4,{'w':2}]])
sasnet.set_node_attributes(G,{1:10,2:20,3:30,4:40},name='x',inplace=True)
```

#use apply to fillna >>> sasnet.apply(G, edge_attr_handler=lambda data:{'w':0 if 'w' not in data else data['w']},inplace=True) >>> assert G.edges[(1,2)]['w'] == 0 >>> print(G.nodes(data=True)) #use apply to negate an attribute >>> sasnet.apply(G, node_attr_handler=lambda data:{'x': -data['x']},inplace=True) >>> assert G.nodes[1]['x']== -10

approximate_diameter

```
approximate_diameter(edge_weight=None)
```

Computes an approximation of the graph's diameter.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted approximate diameter is computed.

Returns

float
Returns the approximate diameter.

articulation_point_count

```
articulation_point_count()
```

Computes a graph's articulation point count.

Returns

int
Returns the number of articulation points.

articulation_points

```
articulation_points()
```

Finds the articulation points (cut points) of a graph.

Returns

list
Returns a list of articulation points in the graph.

assortativity_degree

```
assortativity_degree(source_direction='out', target_direction='in',
edge_weight=None)
```

Computes a graph's degree assortativity.

Parameters

source_direction : {'in', 'out'}
Specifies the degree type (in-degree or out-degree) for the source node in a directed graph.

target_direction : {'in', 'out'}
Specifies the degree type (in-degree or out-degree) for the target node in a directed graph.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted assortativity degree is computed.

Returns

float
Returns the degree assortativity.

assortativity_nominal

```
assortativity_nominal(node_attr)
```

Computes a graph's nominal assortativity.

Parameters

node_attr : str
Specifies the name of the nominal node attribute to use.

Returns

float
Returns the nominal assortativity.

assortativity_numeric

```
assortativity_numeric(node_attr)
```

Computes a graph's numeric assortativity.

Parameters

node_attr : str
Specifies the name of the numeric node attribute to use.

Returns

float
Returns the numeric assortativity.

average_shortest_path

```
average_shortest_path(edge_weight=None)
```

Computes a graph's average shortest path.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted average shortest path is computed.

Returns

float
Returns the average shortest path.

biconcomp_distribution_edges

```
biconcomp_distribution_edges(bins=None)
```

Computes the distribution of the number of edges in each biconnected components by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is int (an integer), it specifies the number of equal-width bins. When *bins* is a list, it specifies bin ranges and should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is str (a string), it specifies the method to use for calculating the bin ranges. For example, 'stone','auto','doane','fd','rice','scott','sqrt','sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of biconnected components whose edge count is within the range of the bin.

biconcomp_distribution_nodes

```
biconcomp_distribution_nodes(bins=None)
```

Computes the distribution of the number of nodes in each biconnected components by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is int (an integer), it specifies the number of equal-width bins. When *bins* is a list, it specifies bin ranges and should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is str (a string), it specifies the method to use for calculating the bin ranges. For example, 'stone','auto','doane','fd','rice','scott','sqrt','sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of biconnected components whose node count is within the range of the bin.

biconcomp_size_edges

```
biconcomp_size_edges()
```

Computes the number of edges within each biconnected component.

Returns

dict
Returns a dictionary in which the keys are biconnected component IDs and the values are the number of edges within each component.

biconcomp_size_nodes

```
biconcomp_size_nodes()
```

Computes the number of nodes within each biconnected component.

Returns

dict
Returns a dictionary in which the keys are biconnected component IDs and the values are the number of nodes within each component.

biconnected_component_count

```
biconnected_component_count()
```

Computes a graph's biconnected component count.

Returns

int
Returns the number of biconnected components.

biconnected_components

```
biconnected_components(order_by=None, *, data=True, ascending=False)
```

Computes the graph's biconnected components for an undirected graph.

Parameters

order_by : 'node' or 'edge' or None, optional
Specifies the parameter that is used to sort the components by the number of nodes or edges within each component. The default is None, which means that the components are not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Returns the result as a generator of biconnected component subgraphs. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

block_cut_tree

```
block_cut_tree(*, return_nodes_map=False)
```

Computes the block-cut tree of a graph.

Parameters

return_nodes_map : bool, optional
Specifies whether or not to also compute the block-cut tree node map. The default is False.

Returns

SAS graph object
Returns a graph that represents the block-cut tree. When *return_nodes_map = True*, the output graph has the *nodes_map* attribute, which contains a dictionary that maps its nodes to a list of nodes in the original graph. Note: If a block is empty (that is, the corresponding biconnected component contains only articulation points), that block does not appear as a key in the *nodes_map* dictionary.

centrality_betweenness

```
centrality_betweenness(edge_weight=None, sample_percent=100, *,
inplace=False, data=True)
```

Computes weighted or unweighted betweenness centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that unweighted betweenness centrality is computed.

sample_percent : int, optional
Specifies the percentage of source nodes to sample for the approximate betweenness calculation. This parameter value should be in the range (0, 100]. The default is 100.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the node and edge attributes *between_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_closeness

```
centrality_closeness(edge_weight=None, nopath='diameter', *, inplace=False,
data=True)
```

Computes weighted or unweighted closeness centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted closeness centrality is computed.

nopath : {'diameter', 'harmonic', or 'zero'}, optional
Specifies a method to use for accounting for the shortest path distance between two nodes when a path does not exist. The valid values are as follows: - 'diameter' uses the graph diameter (plus one) as the shortest path distance between disconnected nodes. - 'harmonic' uses the harmonic formula for calculating closeness centrality. - 'zero' uses zero as the shortest path distance between disconnected nodes.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the closeness centrality node attributes *centr_close_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_degree

```
centrality_degree(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted degree centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted degree centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the degree centrality node attributes *centr_degree_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_eigenvector

```
centrality_eigenvector(edge_weight=None, maxiters=10000, *, inplace=False,
data=True)
```

Computes weighted or unweighted eigenvector centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted eigenvector centrality is computed.

maxiters : int, optional
Specifies the maximum number of iterations in order to limit the amount of computation time that is spent when convergence is slow. The default is 10,000.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the degree centrality node attributes *centr_eigen_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_hub_authority

```
centrality_hub_authority(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted hub and authority centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted hub and authority centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the hub and authority centrality node attributes *centr_hub_** and *centr_auth_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_influence

```
centrality_influence(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted influence centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted influence centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the first- and second-order influence centrality node attributes *centr_influence1_** and *centr_influence2_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_pagerank

```
centrality_pagerank(edge_weight=None, alpha=0.85, tolerance=1e-09, *,
inplace=False, data=True)
```

Computes weighted or unweighted PageRank centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted PageRank centrality is computed.

alpha : float, optional
Specifies the damping parameter for the PageRank centrality (or algorithm). The default is 0.85.

tolerance : float, optional
Specifies the tolerance parameter for the PageRank centrality (or algorithm). The default is 1E-9.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the PageRank centrality node attributes *centr_pagerank_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

clear

```
clear(*args, **kwargs)
```

Removes all nodes and edges from the graph.

clear_edges

```
clear_edges(*args, **kwargs)
```

Removes all edges from the graph but retains nodes.

concomp_distribution_edges

```
concomp_distribution_edges(bins=None, *, strongly=True)
```

Computes the distribution of the number of edges in each connected component by using *numpy.histogram*.

Parameters

bins : int or list or str, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is an integer (int), it represents the number of equal-width bins. When *bins* is a list, it specifies bin ranges and it should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)*

(inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is a string (str), it specifies the method to be used for calculating the bin ranges. For example, 'stone', 'auto', 'doane', 'fd', 'rice', 'scott', 'sqrt', 'sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of components whose edge count within the range of the bin.

concomp_distribution_nodes

```
concomp_distribution_nodes(bins=None, *, strongly=True)
```

Computes the distribution of the number of nodes in each connected component by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is an integer (int), it represents the number of equal-width bins. When *bins* is a list, it specifies bin ranges and it should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is a string (str), it specifies the method to use for calculating the bin ranges. For example, 'stone', 'auto', 'doane', 'fd', 'rice', 'scott', 'sqrt', 'sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of components whose node count is within the range of the bin.

concomp_size_edges

```
concomp_size_edges(*, strongly=True)
```

Computes the number of edges within each connected component.

Parameters

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary in which the keys are connected component IDs and the values are the number of edges within each component.

concomp_size_nodes

```
concomp_size_nodes(*, strongly=True)
```

Computes the number of nodes within each connected component.

Parameters

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary in which the keys are connected component IDs and the values are the number of nodes within each component.

connected_component_count

```
connected_component_count()
```

Computes a graph's connected component count.

Returns

int
Returns the number of connected components.

connected_components

```
connected_components(order_by=None, *, strongly=True, data=True,
ascending=False)
```

Computes the connected components of a graph.

Parameters

order_by : 'node' or 'edge' or None, optional
Specifies whether to sort the components by the number of nodes or edges within each component. The default is None, which means that the components are not sorted.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one connected component. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

copy

```
copy(immutable=None, *, data=True)
```

Returns a copy of the input graph.

Parameters

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, meaning that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns an independent copy of the input.

core_decomposition

```
core_decomposition(maxtime=None, *, data=True)
```

Computes a core decomposition of the graph.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time for core decomposition to spend. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one k-core in increasing core number. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

cycle_count

```
cycle_count(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Counts the number of cycles in a graph.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Returns

int
Returns a count of cycles in the graph.

cycle_length

```
cycle_length(edge_weight=None)
```

Returns the length of the cycle.

Parameters

edge_weight : str, optional
Specifies the name of an edge attribute to use as the weight of edges. By default, edges are considered unweighted (weight = 1).

Returns

number
Returns the total length of the cycle edges.

degree

```
degree(n=None, edge_weight=None)
```

Returns the degree of nodes.

Parameters

n : single node or container of nodes or None, optional
Specifies which nodes to include. The default is None, which means that the degree of every node is returned.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None.

Returns

int or dictionary
Returns the degree of specified node(s).

density

```
density()
```

Computes the density of the graph.

Returns

float
Returns the density of the graph.

diameter

```
diameter(edge_weight=None)
```

Computes a graph's diameter.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted diameter is computed.

Returns

int or float
Returns the diameter.

drop_edge_attributes

```
drop_edge_attributes(attributes_subset=None, edges_subset=None,
immutable=None, *, inplace=False)
```

Removes edge attributes if they exist.

Parameters

attributes_subset : str or list, optional
Specifies the edge attribute or a list of the edge attributes to remove; for example, 'weight' or ['weight','color']. By default, all edge attributes are removed.

edges_subset : list or iterable of edges, optional
Specifies the edges to remove attributes from; for example, [(1,2),(2,3)]. By default the attributes for all edges are removed.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by removing the specified edge attributes. If the *inplace* parameter value is False, a copy of the input graph with the modified edge attributes is returned.

drop_node_attributes

```
drop_node_attributes(attributes_subset=None, nodes_subset=None,
immutable=None, *, inplace=False)
```

Removes node attributes if they exist.

Parameters

attributes_subset : str or list, optional
Specifies the node attribute or list of node attributes to remove; for example, ['weight','color']. By default, all node attributes are removed.

nodes_subset : list or iterable of nodes, optional
Specifies the nodes to drop attributes from; for example, [1,2,3] or (1,2,3) or [1] or (1,). By default, the attributes for all nodes are removed.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by removing the specified node attributes. If the *inplace* parameter value is False, a copy of the input graph with the modified node attributes is returned.

enumerate_cycle_edges

```
enumerate_cycle_edges(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Enumerates the cycles in a graph and returns the edges of the cycles.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Yields

generator
Yields a generator of cycles, each of which is given as a list of the cycles' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_cycle_nodes

```
enumerate_cycle_nodes(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Enumerates the cycles in a graph and returns the nodes of the cycles.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Yields

generator
Yields a generator of cycles, each of which is given as a list of the cycles' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_cycles

```
enumerate_cycles(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None, *,
data=True)
```

Enumerates the cycles in a graph and returns the cycle graph objects.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one cycle of the same type as the input graph. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_path_edges

```
enumerate_path_edges(source=None, sink=None, minlength=1,
maxlength=None, edge_weight=None, node_weight=None, minedgewt=None,
maxedgewt=None, minnodewt=None, maxnodewt=None, maxtime=None)
```

Enumerates the paths in a graph and returns the edges of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

Yields

generator
Yields a generator of Path objects, each of which is given as a list of the paths' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_path_nodes

```
enumerate_path_nodes(source=None, sink=None, minlength=1,
maxlength=None, edge_weight=None, node_weight=None, minedgewt=None,
maxedgewt=None, minnodewt=None, maxnodewt=None, maxtime=None)
```

Enumerates the paths in a graph and returns the nodes of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

Yields

generator
Yields a generator of Path objects, each of which is given as a list of the paths' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_paths

```
enumerate_paths(source=None, sink=None, minlength=1, maxlength=None,
edge_weight=None, node_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, *, data=True)
```

Enumerates the paths in a graph and returns the Path graph objects.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one Path object. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_sequence_shortest_paths

```
enumerate_sequence_shortest_paths(sequence, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None, *, data=True)
```

Enumerates the shortest paths in a graph that visit the indicated sequence nodes in order.

Parameters

sequence : iterable of nodes
Specifies the order in which to visit the required nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Path objects; each object represents a subpath between a sequence of nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_path_edges

```
enumerate_shortest_path_edges(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None)
```

Enumerates the shortest paths in a graph and returns the edges of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

Yields

generator
Yields a generator of paths; each path is given as a list of the paths' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_path_nodes

```
enumerate_shortest_path_nodes(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None)
```

Enumerates the shortest paths in a graph and returns the nodes of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

Yields

generator
Yields a generator of paths; each path is given as a list of the paths' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_paths

```
enumerate_shortest_paths(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None, *, data=True)
```

Enumerates the shortest paths in a graph and returns the Path graph objects.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Path objects; each object represents one shortest path. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

get_meta_graph

```
get_meta_graph(node_type, edge_type=None, immutable=None)
```

Constructs a metagraph from the input graph.

Parameters

node_type : str
Specifies the node attribute to consider in constructing the metagraph.

edge_type : str, optional
Specifies the edge attribute to consider in constructing the metagraph. By default, all connections between nodes are considered the same.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

Returns

SAS graph object
Returns the metagraph, which has an edge attribute count and determines the number of unique connections between pairs of nodes.

get_nlargest_biconcomps

```
get_nlargest_biconcomps(n, order_by='node', *, data=True, ascending=False)
```

Returns a list of up to *n* largest biconnected components.

Parameters

n : int
Specifies the number of graphs to return.

order_by : 'node' or 'edge' or None
Specifies whether to sort the biconnected components by the number of nodes or edges within those components. The default is None, which means that the biconnected components are not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. When the value is True, the smallest biconnected components are returned instead. The default is False.

Returns

list
Returns a list of n largest or smallest biconnected components of a graph.

get_nlargest_concomps

```
get_nlargest_concomps(n, order_by='node', *, strongly=True, data=True,
ascending=False)
```

Returns a list of up to *n* largest connected (with highest number of nodes) components of a graph.

Parameters

n : int
Specifies the number of graphs to return.

order_by : 'node' or 'edge'
Specifies whether to sort the component by number of nodes or edges within that component. The default is None, which means that the result is not sorted.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Returns

list
Returns a list of the n largest connected components of a graph.

has_edge_attribute

```
has_edge_attribute(name, *, all_edges=False)
```

Checks the edges of the input graph for the specified edge attribute.

Parameters

name : str
Specifies the name of the edge attribute.

all_edges : bool, optional
Specifies whether or not to check for the existence of the edge attribute in all or any of the edges. The default is False.

Returns

bool
If the all_edges parameter value is True, the value True is returned if that edge attribute exists in all edges; otherwise the value False is returned. If the all_edges parameter value is False, the value True is returned if at least one edge in the graph has that specified edge attribute.

has_node_attribute

```
has_node_attribute(name, *, all_nodes=False)
```

Checks the nodes of the input graph for the specified node attribute.

Parameters

name : str
Specifies the name of the node attribute.

all_nodes : bool, optional
Specifies whether or not to check for the existence of the node attribute in all or any of the nodes. The default is False.

Returns

bool
If the all_nodes parameter value is True, the value True is returned if that node attribute exists in all node; otherwise the value False is returned. If the

all_nodes parameter value is False, the value True is returned if at least one node in the graph has that specified node attribute.

in_degree

```
in_degree(n=None, edge_weight=None)
```

Returns the in-degree of nodes.

Parameters

n : single node or container of nodes or None, optional
Specifies which nodes to include. The default is None, which means that the in-degree of all nodes is returned.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None.

Returns

numeric or dictionary
Returns the in-degree of the node if a single node is passed, or returns a dictionary with nodes as keys and their in-degrees as values if None or a list of nodes is passed.

intersection

```
intersection(*args, **kwargs)
```

Returns the intersection of input graphs by finding the intersections of their nodes and edges.

The node attributes and edge attributes are aggregated using the rules that are defined by the *nodes_attrs_agg* and *edges_attrs_agg* parameters.

Parameters

rhs : SAS graph object
Specifies the other graph, of the same type as *self*, with which to intersect.

nodes_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate node attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps node attributes to functions. The default is None.

edges_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate edge attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps edge attributes to functions. The default is None.

Returns

SAS graph object
Returns a graph that is constructed by the intersection of the two graphs.

is_biconnected

```
is_biconnected()
```

Specifies whether or not the input graph is biconnected.

Returns

bool
Indicates whether the input graph is biconnected (True) or not (False).

is_bipartite

```
is_bipartite()
```

Determines whether the input graph is bipartite.

Returns

bool
Indicates whether the input graph is bipartite.

is_connected

```
is_connected()
```

Specifies whether or not the input graph is connected.

Returns

bool
Indicates whether the input graph is connected (True) or not (False).

is_cycle

```
is_cycle()
```

Determines whether the input graph is a simple cycle.

Returns

bool
Indicates whether the input graph is a simple cycle (True) or not (False).

is_dag

```
is_dag()
```

Determines whether the input graph is a directed acyclic graph (DAG).

Returns

bool
Indicates whether the input graph is a directed acyclic graph (DAG).

is_directed

```
is_directed()
```

Specifies whether or not the graph is directed.

Returns

bool
Indicates whether or not the graph is directed.

is_hamiltonian

```
is_hamiltonian(maxtime=None)
```

Determines whether the graph is Hamiltonian (has a Hamiltonian cycle) or not.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time to determine whether or not the graph is Hamiltonian. The default is None.

Returns

bool or None
Indicates whether or not the input graph is determined to be Hamiltonian. Returns None if no determination is made.

is_immutable

```
is_immutable()
```

Specifies whether or not the graph is immutable.

Returns

bool
Indicates whether or not the graph is immutable.

is_multigraph

```
is_multigraph()
```

Specifies whether or not the graph allows multiedges.

Returns

bool
Indicates whether or not the graph allows multiedges.

is_path

```
is_path()
```

Determines whether the input graph is a simple path.

Returns

bool
Indicates whether the input graph is a simple path (True) or not (False).

is_strongly_connected

```
is_strongly_connected()
```

Specifies whether the input graph is strongly connected.

Returns

bool
Indicates whether the input graph is strongly connected (True) or not (False).

is_weakly_connected

```
is_weakly_connected()
```

Specifies whether the input graph is weakly connected.

Returns

bool
Indicates whether the input graph is weakly connected (True) or not (False).

label_propagation

```
label_propagation(edge_weight=None, maxiters=100, tolerance=0.05, *,
data=True)
```

Computes the communities of the graph by using the label propagation algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of label changes for all nodes in the graph is less than the value. The default is 0.05.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one community. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

label_propagation_nodes

```
label_propagation_nodes(edge_weight=None, maxiters=100, tolerance=0.05, *,
data=True, inplace=False)
```

Computes community assignments for all nodes in the graph by using the label propagation algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of label changes for all nodes in the graph is less than the value. The default is 0.05.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

inplace : bool, optional
Specifies whether to add the community values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains the node attribute *community*, which denotes the community assignment for each node. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

leaf_node_count

```
leaf_node_count()
```

Computes a graph's leaf node count.

Returns

int
Returns the number of leaf nodes.

local_transitivity

```
local_transitivity(*, inplace=False)
```

Computes the local transitivity (clustering coefficient) and stores the results as node attributes on the graph.

Parameters

inplace : bool, optional
Specifies whether to add the transitivity values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains node attribute 'transitivity'. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

louvain

```
louvain(*, edge_weight=None, resolution=1, tolerance=0.001, maxiters=100,
data=True)
```

Computes the communities of the graph by using the Louvain algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

resolution : float, optional
Specifies the factor in the modularity equation that affects the expected number of links between two nodes. A higher value produces more communities. The default is 1.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of modularity gain between two consecutive iterations is less than the value. The default is 0.001.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one community. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

louvain_nodes

```
louvain_nodes(*, edge_weight=None, resolution=1, tolerance=0.001,
maxiters=100, data=True, inplace=False)
```

Computes community assignments for all nodes in the graph by using the Louvain algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

resolution : float, optional
Specifies the factor in the modularity equation that affects the expected number of links between two nodes. A higher value produces more communities. The default is 1.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of modularity gain between two consecutive iterations is less than the value. The default is 0.001.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

inplace : bool, optional
Specifies whether to add the community values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains the node attribute *community*, which denotes the community assignment for each node. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

maxflow

```
maxflow(source, sink, capacity_attr='upper', *, inplace=False, data=True)
```

Calculates the maximum flow from a source node to a sink node in a graph.

Parameters

source : number or str
Specifies the source node for the maximum flow.

sink : number or str
Specifies the sink node for the maximum flow.

capacity_attr : str, optional
Specifies the name of the edge attribute that contains the capacity of the edge. The default is 'upper'.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute *flow*, which contains the flow values. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, the returned graph contains only the flow edges.

maximal_cliques

```
maximal_cliques(maxcliques='ALL', node_weight=None, edge_weight=None,
minsize=1, maxsize=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, order_by=None, *,
data=True, ascending=False)
```

Enumerates all maximal cliques of the graph.

Parameters

maxcliques : int or 'ALL', optional
Specifies the maximum number of enumerated cliques to return. The default is 'ALL'.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

minsize : int, optional
Specifies the minimum number of nodes in a clique. The default is 1.

maxsize : int or None, optional
Specifies the maximum number of nodes in a clique. The default is None.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a clique. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a clique. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a clique. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a clique. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds for enumeration to spend. The default is None.

order_by : 'node' or None, optional
Specifies whether to sort the cliques by the number of nodes within each clique. The default is None, which means that the result is not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one maximal clique. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

min_cut

```
min_cut(edge_weight=None, *, inplace=False, data=True)
```

Calculates a minimum cut of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted minimum cut is calculated.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute 'mincut', which indicates the edges to include in the minimum cut set, and the node attribute 'mincut_partition', which indicates the node partition of the calculated minimum cut. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, a modified copy of the graph is returned.

min_s_t_cut

```
min_s_t_cut(source, sink, edge_weight=None, *, inplace=False, data=True)
```

Calculates a minimum s-t cut of a graph.

Parameters

source : number or str
Specifies the source node (s) for the minimum s-t cut.

sink : number or str
Specifies the sink node (t) for the minimum s-t cut.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted minimum s-t cut is calculated.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute *s_t_cut*, which indicates the edges to include in the minimum s-t cut set; and the node attribute *s_t_cut_partition*, which indicates the node partition of the calculated minimum s-t cut. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, a modified copy of the graph is returned.

mincostflow

```
mincostflow(demand_lower_attr='lower', demand_upper_attr='upper',
capacity_lower_attr='lower', capacity_upper_attr='upper', cost_attr='weight',
maxtime=None, *, inplace=False, data=True)
```

Calculates the minimum-cost network flow for the graph.

Parameters

demand_lower_attr : str, optional
Specifies the name of the node attribute that contains the lower bound of the node supply. The default is 'lower'.

demand_upper_attr : str, optional
Specifies the name of the node attribute that contains the upper bound of the node supply. The default is 'upper'.

capacity_lower_attr : str, optional
Specifies the name of the edge attribute that contains the capacity lower bound of the edge. The default is 'lower'.

capacity_upper_attr : str, optional
Specifies the name of the edge attribute that contains the capacity upper bound of the edge. The default is 'upper'.

cost_attr : str, optional
Specifies the name of the edge attribute that contains the edge cost. The default is 'weight'.

maxtime : float or None, optional
Specifies the maximum amount of time to allow for computing the minimum-cost network flow. The default is None.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attributes *flow* and *reduced_cost*, which contain the flow and reduced cost values, respectively; and the node attribute *dual*, which contains the optimal dual value. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned.

minimum_spanning_tree

```
minimum_spanning_tree(edge_weight=None, source=None, *, inplace=False,
data=True)
```

Calculates a minimum spanning tree (MST) or forest of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

source : number or str or None
Specifies the source node for a directed MST. The default is None.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute 'mst', which indicates the edges to include in the spanning tree or forest. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, the returned graph contains only the spanning tree or forest edges.

minimum_weight_full_matching

```
minimum_weight_full_matching(edge_weight=None, *, data=True)
```

Returns a minimum-weight full matching of the bipartite graph *self*.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns a directed graph whose edges identify the assignment mapping in the minimum-weight full matching solution. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

neighbors

```
neighbors(n)
```

Returns an iterator over all neighbors of node n.

Parameters

n : node
Specifies a node in the graph.

Returns

iterator
Returns an iterator over all neighbors of node n.

node_clique_numbers

```
node_clique_numbers(*, inplace=False, data=True)
```

Computes node clique numbers and stores the results as attributes on the graph.

Parameters

inplace : bool, optional
Specifies whether to add the clique number values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the value of the *inplace* parameter is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph or modifies the existing graph to contain the node attribute *clique_number*, which includes the clique number of each node. The graph g has the attribute g.solution_summary, which contains information about the results of the algorithm.

number_of_edges

```
number_of_edges(u=None, v=None)
```

Returns the number of edges in the graph.

When you specify *u* and *v*, the number of edges from *u* to *v* is returned. When you specify *u* but *v* is None, the number of connections from *u* to other nodes (out-degree of node *u*) is returned. When you specify *v* but *u* is None, the number of connections from *v* to other nodes (in-degree of node *v*) is returned.

Parameters

u : numeric or str or None, optional
Specifies the *from* node. The default is None.

v : numeric or str or None, optional
Specifies the *to* node. The default is None.

Returns

int
Returns the number of edges in the graph. When you specify *u* and *v*, the number of edges from *u* to *v* is returned. When you specify *u* but *v* is None, the number of connections from *u* to other nodes(out-degree of node *u*) is returned. When you specify *v* but *u* is None, the number of connections from *v* to other nodes(in-degree of node *v*) is returned.

number_of_nodes

```
number_of_nodes()
```

Returns the number of nodes in the graph.

Returns

int
Returns the number of nodes in the graph.

out_degree

```
out_degree(n=None, edge_weight=None)
```

Returns the out-degree of nodes.

Parameters

n : single node or container of nodes or None, optional
Specifies which nodes to include. The default is None, which means that the out-degree of all nodes is returned.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None.

Returns

numeric or dictionary
Returns the out-degree of the node if a single node is passed, or returns a dictionary with nodes as keys and their out-degrees as values if None or a list of nodes is passed.

predecessors

```
predecessors(n)
```

Returns an iterator over all predecessor nodes of n.

Parameters

n : node
Specifies a node in the graph.

Returns

iterator
Returns an iterator over predecessors of node n.

projected_graph

```
projected_graph(nodes, neighbors=None, directed_method=None,
_measure=None, *, multigraph=False, data=True)
```

Computes a projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in self other than *nodes*.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used. When the *directed_method* parameter is specified for an undirected graph, a ValueError is raised.

multigraph : bool, optional
Specifies whether or not to return a multigraph. When the value is True, the algorithm returns a multigraph in which the multiple edges represent multiple shared neighbors. The edge attribute 'neighbor' stores the label of the shared neighbor. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

projected_graph_adamic_adar

```
projected_graph_adamic_adar(nodes, neighbors=None, directed_method=None,
*, data=True)
```

Computes an Adamic-Adar projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than *nodes*.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'adamic_adar' edge attribute.

projected_graph_common_neighbors

```
projected_graph_common_neighbors(nodes, neighbors=None,
directed_method=None, *, data=True)
```

Computes a common neighbors projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'common_neighbors' edge attribute.

projected_graph_cosine

```
projected_graph_cosine(nodes, edge_weight=None, neighbors=None,
directed_method=None, *, data=True)
```

Computes a cosine projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'cosine' edge attribute.

projected_graph_dice

```
projected_graph_dice(nodes, neighbors=None, directed_method=None, *,
data=True)
```

Computes a Sørensen-Dice projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the subset of nodes that are specified by the *neighbors* parameter. The strength of association in the calculated projection is represented by the 'dice' edge attribute.

projected_graph_jaccard

```
projected_graph_jaccard(nodes, neighbors=None, directed_method=None, *,
data=True)
```

Computes a Jaccard projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'jaccard' edge attribute.

query

```
query(q, expr=None, _node_attr_to_ignore=None, _edge_attr_to_ignore=None, *,
induced=False, break_symmetry=False, _generate_output=True,
_orbits_only=False, **kwargs)
```

Queries for the subgraph isomorphisms of q within g.

Parameters

q : graph
Specifies the query graph.

expr : _Expression or None, optional
Specifies additional logic to filter the query results. The default is None.

induced : bool, optional
Specifies whether or not to return only node-induced matches. The default is False.

break_symmetry : bool, optional
Specifies whether or not to break symmetry and eliminate automorphic relationships between matches. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one isomorphic mapping from *q* to *self*. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

reach_graph

```
reach_graph(source, max_reach=1, *, data=True)
```

Computes the reach graph for the source or set of sources.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to compute the reach network.

max_reach : int, optional
Specifies the number of steps (or hops) to traverse from the source nodes. The default is 1.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns a graph that represents the reach network. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

reach_graph_per_source

```
reach_graph_per_source(source, max_reach=1, *, data=True)
```

Computes a separate reach graph for each given source node.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to compute the reach network.

max_reach : int, optional
Specifies the number of steps (or hops) to traverse from the source nodes. The default is 1.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one reach network. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

remove_edge

```
remove_edge(*args, **kwargs)
```

Removes a single edge from the graph.

Parameters

u : numeric or str
Specifies the *from* node.

v : numeric or str
Specifies the *to* node.

Examples

```
G.remove_edge(1,2)
```

remove_edges_from

```
remove_edges_from(*args, **kwargs)
```

Removes specified edges from the graph.

Parameters

edges : an iterable container of edges
Specifies the edges to remove from the graph.

remove_isolated_nodes

```
remove_isolated_nodes(*args, **kwargs)
```

Removes isolated nodes from a graph.

Parameters

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the input graph. The default is False.

Returns

SAS graph object or None
Returns the resulting graph after isolated nodes are removed, or returns nothing if the *inplace* parameter value is True.

remove_node

```
remove_node(*args, **kwargs)
```

Removes a single node from the graph.

Parameters

node : numeric or str
Specifies the node to remove from the graph.

remove_nodes_from

```
remove_nodes_from(*args, **kwargs)
```

Removes specified nodes from the graph.

If a specified node does not exist in the graph, it is silently ignored.

Parameters

nodes : an iterable container of nodes
Specifies the nodes to remove from the graph.

reverse

```
reverse(*, data=True)
```

Returns a copy of a directed graph with reversed edges.

Parameters

data : bool, optional
Specifies whether or not to retain attributes. The default is True, meaning that the node and edge attributes of the graph are retained.

Returns

directed graph object
Returns a directed graph that holds the reversed edges.

set_edge_attributes

```
set_edge_attributes(values, name=None, immutable=None, *, inplace=False)
```

Sets edge attributes by using a given value or dictionary of values.

Parameters

values : scalar or dict-like or Pandas DataFrame
Specifies what the edge attribute should be set to. When this parameter is set to a Pandas DataFrame, it is expected to have from, to, and edge_key (only for multigraphs) columns, and the other columns are treated as edge attributes to add. When *values* is a dictionary, the keys are edges and the values are either edge attribute values or dictionaries of attribute name-value pairs. For multigraphs, the edge tuples must be of the form (u, v, key). For simple graphs, the keys must be tuples of the form (u, v). When this parameter is not set to a dictionary, it is treated as a single attribute value that is then applied to every edge in *self*. The attribute name is specified by the *name* parameter.

name : str or None, optional
Specifies the name of the edge attribute to set if the *values* parameter is set to a scalar. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is False, a copy of the input graph with the new edge attributes is returned. If the *inplace* parameter value is True, the graph is modified by setting the specified edge attributes.

set_node_attributes

```
set_node_attributes(values, name=None, immutable=None, *, inplace=False)
```

Sets node attributes from a given value or dictionary of values.

Parameters

values : scalar or dict-like or Pandas DataFrame
Specifies what the node attribute should be set to. When *values* is not a dictionary, it is treated as a single attribute value that is then applied to every node in *self*. This means that if you provide a mutable object, such as a list, updates to that object are reflected in the node attribute for every node. The attribute name is specified by the *name* parameter. When *values* is a dictionary, the keys are nodes and the values are either node attribute values or dictionaries of attribute name-value pairs.

name : str or None, optional
Specifies the name of the node attribute to set if the *values* parameter is set to a scalar. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not the graph is modified in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by setting the specified node attributes. If the *inplace* parameter value is False, a copy of the input graph with the new node attributes is returned.

shortest_path_distances

```
shortest_path_distances(source=None, sink=None, edge_weight=None,
minedgewt=None, maxedgewt=None)
```

Calculates the shortest path distances in a graph.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) and restricts enumeration to the shortest paths that contain the specified source node(s). The default is None.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) and restricts enumeration to the shortest paths that contain the specified sink node(s). The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

Returns

dict
Returns a dictionary where each key is a (source, sink) tuple and its value is the corresponding shortest source-sink path distance.

similarity_adamic_adar

```
similarity_adamic_adar(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Adamic-Adar node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_common_neighbors

```
similarity_common_neighbors(source, sink=None, top_k=None, bottom_k=None,
*, exclude_source=False, exclude_existing_neighbors=False)
```

Computes common neighbors node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_cosine

```
similarity_cosine(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes cosine node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_dice

```
similarity_dice(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Sørensen-Dice node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_graph

```
similarity_graph(measure, min_score=2.220446049250313e-16, max_score=inf,
*, exclude_source=False, exclude_existing_neighbors=False, data=True,
**kwargs)
```

Computes a node similarity graph.

Parameters

measure : str, one of {'adamic_adar', 'common_neighbors', 'cosine', 'dice', 'jaccard'}
Specifies the metric to be used to compute the node similarity.

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the edge attribute that corresponds to the measure name. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_adamic_adar

```
similarity_graph_adamic_adar(min_score=2.220446049250313e-16,
max_score=inf, *, exclude_source=False, exclude_existing_neighbors=False,
data=True)
```

Computes an Adamic-Adar node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'adamic_adar' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_common_neighbors

```
similarity_graph_common_neighbors(min_score=2.220446049250313e-16,
max_score=inf, *, exclude_source=False, exclude_existing_neighbors=False,
data=True)
```

Computes a common neighbors node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'common_neighbors' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_cosine

```
similarity_graph_cosine(edge_weight=None,
min_score=2.220446049250313e-16, max_score=inf, *, exclude_source=False,
exclude_existing_neighbors=False, data=True)
```

Computes a cosine node similarity graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'cosine' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_dice

```
similarity_graph_dice(min_score=2.220446049250313e-16, max_score=inf, *,
exclude_source=False, exclude_existing_neighbors=False, data=True)
```

Computes a Sørensen-Dice node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the *dice* edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_jaccard

```
similarity_graph_jaccard(min_score=2.220446049250313e-16, max_score=inf, *,
exclude_source=False, exclude_existing_neighbors=False, data=True)
```

Computes a Jaccard node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'jaccard' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_jaccard

```
similarity_jaccard(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Jaccard node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

singleton_node_count

```
singleton_node_count()
```

Computes a graph's singleton node count.

Returns

int
Returns the number of singleton nodes.

successors

```
successors(n)
```

Returns an iterator over all successor nodes of n.

Parameters

n : node
Specifies a node in the graph.

Returns

iterator
Returns an iterator over successors of node n.

to_directed

```
to_directed()
```

Returns a directed copy of the input graph.

Returns

SAS graph object
Returns a directed copy of the input graph.

to_immutable

```
to_immutable()
```

Creates an immutable graph from the input graph.

Returns

SAS graph object
Returns an immutable SAS graph object.

to_mutable

```
to_mutable()
```

Creates a mutable graph from the input graph.

Returns

SAS graph object
Returns a mutable SAS graph object.

to_pandas_edges

```
to_pandas_edges(*, data=True, _for_conversion_to_immutable=False)
```

Returns the graph edges as a Pandas DataFrame.

Parameters

data : bool or list, optional
Specifies whether or not to retain attributes. The default is True. When the value is True (or when a list is given), the Pandas DataFrame includes all the edge attributes (or all the listed attributes).

Returns

dataframe
Returns a Pandas DataFrame that includes the graph edges.

to_pandas_nodes

```
to_pandas_nodes(*, data=True, _for_conversion_to_immutable=False)
```

Returns the graph nodes as a Pandas DataFrame.

Parameters

data : bool or list, optional
Specifies whether or not to retain attributes. The default is True. When the value is True (or when a list is given), the Pandas DataFrame includes all the node attributes (or all the listed attributes).

Returns

dataframe
Returns a Pandas DataFrame that includes the graph nodes.

to_undirected

```
to_undirected()
```

Returns a directed copy of the input graph.

Returns

SAS graph object
Returns a directed copy of the input graph.

topological_ordering

```
topological_ordering()
```

Finds a topological ordering of the graph's nodes.

Yields

generator
Yields a generator of nodes in topological order. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

transitive_closure

```
transitive_closure()
```

Calculates the transitive closure of a graph.

Returns

SAS graph object
Returns the transitive closure of the input graph that is returned as a SAS MultiGraph or MultiDiGraph object. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

triangle_count

```
triangle_count()
```

Computes a graph's triangle count.

Returns

int
Returns the number of triangles.

tsp_tour

```
tsp_tour(edge_weight=None, maxtime=None, minobjective=None, *,
inplace=False, use_milp=True, data=True, **options)
```

Calculates a traveling salesman problem (TSP) tour of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted TSP is calculated.

maxtime : float or None, optional
Specifies the maximum amount of time for solving a TSP. The default is None.

minobjective : float, optional
Specifies a minimum value such that when a solution is found whose objective is less than or equal to this value, the algorithm stops.

inplace : bool, optional
Specifies whether to add the TSP order values as attributes to the input graph directly or to create a copy of the graph. The default is False.

use_milp : bool, optional
Specifies whether or not to use a mixed integer linear programming (MILP) solver to solve a TSP. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is False, the returned graph is a Cycle that contains the calculated TSP tour edges. When the *inplace* parameter value is True, the graph is modified in place and None is returned. The modified graph contains the edge attributes *tsp* and *tsp_order* and the node attribute *tsp_order*.

union

```
union(*args, **kwargs)
```

Returns the union of input graphs by finding the unions of their nodes and edges.

The node attributes and edge attributes are aggregated using the rules that are defined by the *nodes_attrs_agg* and *edges_attrs_agg* parameters.

Parameters

rhs : SAS graph object
Specifies the other graph, of the same type as *self*, with which to find the union.

nodes_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate node attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps node attributes to functions. The default is None.

edges_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate edge attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps edge attributes to functions. The default is None.

Returns

SAS graph object
Returns a graph that is constructed by the union of the two graphs.

vrp

```
vrp(capacity, depot, node_demand='demand', edge_cost='weight', minroutes=1,
maxroutes=None, maxtime=None, minobjective=None, *, use_milp=True,
data=True, **options)
```

Solves the vehicle routing problem.

Parameters

capacity : float
Specifies the capacity of each vehicle.

depot : node
Specifies the depot node for the vehicle routing problem.

node_demand : str or dict or iterable
Specifies the quantity of goods to deliver to or pick up from a customer. The default is 'demand'.

edge_cost : str or dict or iterable
Specifies the cost of traversing each edge. The default is 'weight'.

minroutes : int, optional
Specifies the minimum number of routes that are allowed to service demand.

maxroutes : int, optional
Specifies the maximum number of routes that are allowed to service demand.

maxtime : float or None, optional
Specifies the maximum amount of time for solving the vehicle routing problem. The default is None.

minobjective : float, optional
Specifies a minimum value such that when a solution is found whose objective is less than or equal to this value, the algorithm stops.

use_milp : bool, optional
Specifies whether or not to use a mixed integer linear programming (MILP) solver to solve the vehicle routing problem. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Cycle objects; each cycle represents one route. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

vrptw

```
vrptw(capacity, depot, node_demand='demand', edge_cost='weight',
travel_time='weight', timewindow_lower=None, timewindow_upper=None,
service_time=None, minroutes=1, maxroutes=None, maxtime=None, *,
data=True)
```

Solves the vehicle routing problem with time windows.

Parameters

capacity : float
Specifies the capacity of each vehicle.

depot : node
Specifies the depot node for the vehicle routing problem.

node_demand : str or dict or iterable
Specifies the quantity of goods to deliver to or pick up from a customer. The default is 'demand'.

edge_cost : str or dict or iterable
Specifies the cost of traversing each edge.

travel_time : str or dict or iterable
Specifies attributes or values that define the time that it takes to traverse each edge. The default is 'weight'.

timewindow_lower : str or dict or iterable or None, optional
Specifies attributes or values that define the lower time limits within which a delivery vehicle can arrive at the customer nodes. The default is None.

timewindow_upper : str or dict or iterable or None, optional
Specifies attributes or values that define the upper time limits within which a delivery vehicle can arrive at the customer nodes. The default is None.

service_time : str or dict or iterable or None, optional
Specifies attributes or values that define the service time of the nodes. The default is None.

minroutes : int, optional
Specifies the minimum number of routes that are allowed to service demand.

maxroutes : int, optional
Specifies the maximum number of routes that are allowed to service demand.

maxtime : float or None, optional
Specifies the maximum amount of time for solving the vehicle routing problem. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Cycle objects; each cycle represents one route. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

Path

Class for result graphs that are paths.

API: Path

class sasviya.network.classes.path.Path(input_graph=None, source=None, sink=None, data=None, edge_weight=None, *args, **kwargs)

Parameters

input_graph : SAS Graph or None, optional
Specifies the underlying graph for the path. The default is None.

source : node or None, optional
Specifies the path's source node. The default is None.

sink : node or None, optional
Specifies the path's sink node. The default is None.

data : bool or None, optional
Specifies whether or not to include attributes. The default is None. When the value is None or True, the node and edge attributes of the input graph are added to the Path object.

edge_weight : str or None, optional
Specifies the edge attribute to use for calculating the path length. The default is None, which means that unit weights are used.

Attributes

edges
Returns the edges of the graph.

nodes
Returns the nodes of the graph.

sink
Returns the sink node of the path.

source
Returns the source node of the path.

Methods

Method Name	Description
add_core_number_attr	Computes the core numbers and stores the results as node attributes on the graph.
add_edge	Adds a single edge to the graph.
add_edges_from	Adds specified edges to the graph.
add_node	Adds a node to the graph.
add_nodes_from	Adds the specified nodes to the graph.
add_topological_order_attr	Computes a topological ordering and stores the results as node attributes of the graph.
add_weighted_edges_from	Adds weighted edges to the graph.
adjacency	Returns an iterator over (node, adjacency dict) for all nodes.

Method Name	Description
allows_multiedges	Specifies whether or not the graph allows multiedges.
allows_selfloops	Specifies whether or not the graph allows self-loops.
apply	Applies a function to all node or edge attributes and updates them.
approximate_diameter	Computes an approximation of the graph's diameter.
articulation_point_count	Computes a graph's articulation point count.
articulation_points	Finds the articulation points (cut points) of a graph.
assortativity_degree	Computes a graph's degree assortativity.
assortativity_nominal	Computes a graph's nominal assortativity.
assortativity_numeric	Computes a graph's numeric assortativity.
average_shortest_path	Computes a graph's average shortest path.
biconcomp_distribution_edges	Computes the distribution of the number of edges in each biconnected components by using *numpy.histogram*.
biconcomp_distribution_nodes	Computes the distribution of the number of nodes in each biconnected components by using *numpy.histogram*.
biconcomp_size_edges	Computes the number of edges within each biconnected component.
biconcomp_size_nodes	Computes the number of nodes within each biconnected component.
biconnected_component_count	Computes a graph's biconnected component count.
biconnected_components	Computes the graph's biconnected components for an undirected graph.
block_cut_tree	Computes the block-cut tree of a graph.
centrality_betweenness	Computes weighted or unweighted betweenness centrality and stores the results as attributes on the graph.
centrality_closeness	Computes weighted or unweighted closeness centrality and stores the results as attributes on the graph.

Method Name	Description
centrality_degree	Computes weighted or unweighted degree centrality and stores the results as attributes on the graph.
centrality_eigenvector	Computes weighted or unweighted eigenvector centrality and stores the results as attributes on the graph.
centrality_hub_authority	Computes weighted or unweighted hub and authority centrality and stores the results as attributes on the graph.
centrality_influence	Computes weighted or unweighted influence centrality and stores the results as attributes on the graph.
centrality_pagerank	Computes weighted or unweighted PageRank centrality and stores the results as attributes on the graph.
clear	Removes all nodes and edges from the graph.
clear_edges	Removes all edges from the graph but retains nodes.
concomp_distribution_edges	Computes the distribution of the number of edges in each connected component by using *numpy.histogram*.
concomp_distribution_nodes	Computes the distribution of the number of nodes in each connected component by using *numpy.histogram*.
concomp_size_edges	Computes the number of edges within each connected component.
concomp_size_nodes	Computes the number of nodes within each connected component.
connected_component_count	Computes a graph's connected component count.
connected_components	Computes the connected components of a graph.
copy	Returns a copy of the input graph.
core_decomposition	Computes a core decomposition of the graph.
cycle_count	Counts the number of cycles in a graph.
degree	Returns the degree of nodes.

Method Name	Description
density	Computes the density of the graph.
diameter	Computes a graph's diameter.
drop_edge_attributes	Removes edge attributes if they exist.
drop_node_attributes	Removes node attributes if they exist.
enumerate_cycle_edges	Enumerates the cycles in a graph and returns the edges of the cycles.
enumerate_cycle_nodes	Enumerates the cycles in a graph and returns the nodes of the cycles.
enumerate_cycles	Enumerates the cycles in a graph and returns the cycle graph objects.
enumerate_path_edges	Enumerates the paths in a graph and returns the edges of the paths.
enumerate_path_nodes	Enumerates the paths in a graph and returns the nodes of the paths.
enumerate_paths	Enumerates the paths in a graph and returns the Path graph objects.
enumerate_sequence_shortest_paths	Enumerates the shortest paths in a graph that visit the indicated sequence nodes in order.
enumerate_shortest_path_edges	Enumerates the shortest paths in a graph and returns the edges of the paths.
enumerate_shortest_path_nodes	Enumerates the shortest paths in a graph and returns the nodes of the paths.
enumerate_shortest_paths	Enumerates the shortest paths in a graph and returns the Path graph objects.
get_meta_graph	Constructs a metagraph from the input graph.
get_nlargest_biconcomps	Returns a list of up to *n* largest biconnected components.
get_nlargest_concomps	Returns a list of up to *n* largest connected (with highest number of nodes) components of a graph.
has_edge_attribute	Checks the edges of the input graph for the specified edge attribute.
has_node_attribute	Checks the nodes of the input graph for the specified node attribute.

Method Name	Description
in_degree	Returns the in-degree of nodes.
intersection	Returns the intersection of input graphs by finding the intersections of their nodes and edges.
is_biconnected	Specifies whether or not the input graph is biconnected.
is_bipartite	Determines whether the input graph is bipartite.
is_connected	Specifies whether or not the input graph is connected.
is_cycle	Determines whether the input graph is a simple cycle.
is_dag	Determines whether the input graph is a directed acyclic graph (DAG).
is_directed	Specifies whether or not the graph is directed.
is_hamiltonian	Determines whether the graph is Hamiltonian (has a Hamiltonian cycle) or not.
is_immutable	Specifies whether or not the graph is immutable.
is_multigraph	Specifies whether or not the graph allows multiedges.
is_path	Determines whether the input graph is a simple path.
is_strongly_connected	Specifies whether the input graph is strongly connected.
is_weakly_connected	Specifies whether the input graph is weakly connected.
label_propagation	Computes the communities of the graph by using the label propagation algorithm.
label_propagation_nodes	Computes community assignments for all nodes in the graph by using the label propagation algorithm.
leaf_node_count	Computes a graph's leaf node count.
local_transitivity	Computes the local transitivity (clustering coefficient) and stores the results as node attributes on the graph.
louvain	Computes the communities of the graph by using the Louvain algorithm.

Method Name	Description
louvain_nodes	Computes community assignments for all nodes in the graph by using the Louvain algorithm.
maxflow	Calculates the maximum flow from a source node to a sink node in a graph.
maximal_cliques	Enumerates all maximal cliques of the graph.
min_cut	Calculates a minimum cut of a graph.
min_s_t_cut	Calculates a minimum s-t cut of a graph.
mincostflow	Calculates the minimum-cost network flow for the graph.
minimum_spanning_tree	Calculates a minimum spanning tree (MST) or forest of a graph.
minimum_weight_full_matching	Returns a minimum-weight full matching of the bipartite graph *self*.
neighbors	Returns an iterator over all neighbors of node n.
node_clique_numbers	Computes node clique numbers and stores the results as attributes on the graph.
number_of_edges	Returns the number of edges in the graph.
number_of_nodes	Returns the number of nodes in the graph.
out_degree	Returns the out-degree of nodes.
path_length	Returns the total length of the path (edges).
predecessors	Returns an iterator over all predecessor nodes of n.
projected_graph	Computes a projected graph.
projected_graph_adamic_adar	Computes an Adamic-Adar projected graph.
projected_graph_common_neighbors	Computes a common neighbors projected graph.
projected_graph_cosine	Computes a cosine projected graph.
projected_graph_dice	Computes a Sørensen-Dice projected graph.
projected_graph_jaccard	Computes a Jaccard projected graph.
query	Queries for the subgraph isomorphisms of q within g.

Method Name	Description
reach_graph	Computes the reach graph for the source or set of sources.
reach_graph_per_source	Computes a separate reach graph for each given source node.
remove_edge	Removes a single edge from the graph.
remove_edges_from	Removes specified edges from the graph.
remove_isolated_nodes	Removes isolated nodes from a graph.
remove_node	Removes a single node from the graph.
remove_nodes_from	Removes specified nodes from the graph.
reverse	Returns a copy of a directed graph with reversed edges.
set_edge_attributes	Sets edge attributes by using a given value or dictionary of values.
set_node_attributes	Sets node attributes from a given value or dictionary of values.
shortest_path_distances	Calculates the shortest path distances in a graph.
similarity_adamic_adar	Computes Adamic-Adar node similarity.
similarity_common_neighbors	Computes common neighbors node similarity.
similarity_cosine	Computes cosine node similarity.
similarity_dice	Computes Sørensen-Dice node similarity.
similarity_graph	Computes a node similarity graph.
similarity_graph_adamic_adar	Computes an Adamic-Adar node similarity graph.
similarity_graph_common_neighbors	Computes a common neighbors node similarity graph.
similarity_graph_cosine	Computes a cosine node similarity graph.
similarity_graph_dice	Computes a Sørensen-Dice node similarity graph.
similarity_graph_jaccard	Computes a Jaccard node similarity graph.
similarity_jaccard	Computes Jaccard node similarity.

Method Name	Description
singleton_node_count	Computes a graph's singleton node count.
successors	Returns an iterator over all successor nodes of n.
to_directed	Returns a directed copy of the input graph.
to_immutable	Creates an immutable graph from the input graph.
to_mutable	Creates a mutable graph from the input graph.
to_pandas_edges	Returns the graph edges as a Pandas DataFrame.
to_pandas_nodes	Returns the graph nodes as a Pandas DataFrame.
to_undirected	Returns a directed copy of the input graph.
topological_ordering	Finds a topological ordering of the graph's nodes.
transitive_closure	Calculates the transitive closure of a graph.
triangle_count	Computes a graph's triangle count.
tsp_tour	Calculates a traveling salesman problem (TSP) tour of a graph.
union	Returns the union of input graphs by finding the unions of their nodes and edges.
vrp	Solves the vehicle routing problem.
vrptw	Solves the vehicle routing problem with time windows.

add_core_number_attr

```
add_core_number_attr(maxtime=None, *, inplace=False, data=True)
```

Computes the core numbers and stores the results as node attributes on the graph.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time for core decomposition to spend. The default is None.

inplace : bool, optional
Specifies whether to add the core numbers as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph with core numbers that are stored as a node attribute named *core*. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

add_edge

```
add_edge(*args, **kwargs)
```

Adds a single edge to the graph.

Parameters

u : numeric or str
Specifies the *from* node.

v : numeric or str
Specifies the *to* node.

error_on_selfloop : bool or None, optional
Specifies whether or not to raise a ValueError if a self-loop is detected; that is, if (u==v). The default is None, which means that the input edge is silently ignored.

Examples

```
G.add_edge(1,2,weight=3,type='parent')
```

add_edges_from

```
add_edges_from(*args, **kwargs)
```

Adds specified edges to the graph.

Parameters

edges : an iterable container of edges
Specifies the edges to add to the graph.

error_on_selfloop : bool or None, optional
Specifies whether or not to raise a ValueError if a self-loop is detected; that is, if (u==v). The default is None, which means that the input self-loop is silently ignored.

Examples

```
G.add_edges_from([("A","B",{"weight":1,"color":"blue"}),
                      ("C","D",{"weight":2,"color":"red"})])
```

```
G.add_edges_from([['X', 'Y'],["Y","Z"], weight=1,color= "blue")
```

add_node

```
add_node(*args, **kwargs)
```

Adds a node to the graph.

Parameters

node : str or number or tuple
Specifies the ID of the node to add to the graph. If the value is a tuple, the first element should contain the node ID, and the second element is a dictionary that represents the node's attributes.

add_nodes_from

```
add_nodes_from(*args, **kwargs)
```

Adds the specified nodes to the graph.

Parameters

nodes : an iterable container of nodes
Specifies the nodes to add to the graph.

Examples

```
G.add_nodes_from([('X', {"weight":11,"color": "blue"}), ("Y", {"color": "blue"})])
G.add_nodes_from(('X', 'Y'), weight=1,color= "blue")
G.add_nodes_from(['X', 'Y'])
```

add_topological_order_attr

add_topological_order_attr(*, inplace=False, data=True)

Computes a topological ordering and stores the results as node attributes of the graph.

Parameters

inplace : bool, optional
Specifies whether or not to add the topological order values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a directed graph whose topological order is stored as a node attribute named *top_order*. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

add_weighted_edges_from

add_weighted_edges_from(*args, **kwargs)

Adds weighted edges to the graph.

Parameters

edges : an iterable container of edges
Specifies the weighted edges to add to the graph.

weight : str
Specifies the attribute name for the weighted edges to add. The default is 'weight'.

adjacency

```
adjacency()
```

Returns an iterator over (node, adjacency dict) for all nodes.

For directed graphs, only outgoing adjacencies are returned.

Returns

iterator
Returns an iterator over (node, adjacency dictionary) for all nodes in the graph.

allows_multiedges

```
allows_multiedges()
```

Specifies whether or not the graph allows multiedges.

Returns

bool
Indicates whether or not the graph allows multiedges.

allows_selfloops

```
allows_selfloops()
```

Specifies whether or not the graph allows self-loops.

Returns

bool
Indicates whether or not the graph allows self-loops.

apply

apply(node_attr_handler=None, edge_attr_handler=None, immutable=None, *, inplace=False)

Applies a function to all node or edge attributes and updates them.

#use apply to fillna >>> sasnet.apply(G, edge_attr_handler=lambda data:{'w':0 if 'w' not in data else data['w']},inplace=True) >>> assert G.edges[(1,2)]['w'] == 0 >>> print(G.nodes(data=True)) #use apply to negate an attribute >>> sasnet.apply(G, node_attr_handler=lambda data:{'x': -data['x']},inplace=True) >>> assert G.nodes[1]['x']== -10

Parameters

node_attr_handler : function or None, optional
Specifies the function to apply to node attributes. The default is None.

edge_attr_handler : function or None, optional
Specifies the function to apply to edge attributes. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the input graph. The default is False.

Returns

SAS graph object or None.
Returns the resulting graph after the function is applied, or returns nothing if the *inplace* parameter value is True.

Examples

```
import sasviya.network as sasnet
G = sasnet.Graph([[1,2,{}],[1,3,{'w':1}],[1,4,{'w':2}]])
sasnet.set_node_attributes(G,{1:10,2:20,3:30,4:40},name='x',inplace=True)
```

#use apply to fillna >>> sasnet.apply(G, edge_attr_handler=lambda data:{'w':0 if 'w' not in data else data['w']},inplace=True) >>> assert G.edges[(1,2)]['w'] == 0 >>> print(G.nodes(data=True)) #use apply to negate an attribute >>> sasnet.apply(G, node_attr_handler=lambda data:{'x': -data['x']},inplace=True) >>> assert G.nodes[1]['x']== -10

approximate_diameter

```
approximate_diameter(edge_weight=None)
```

Computes an approximation of the graph's diameter.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted approximate diameter is computed.

Returns

float
Returns the approximate diameter.

articulation_point_count

```
articulation_point_count()
```

Computes a graph's articulation point count.

Returns

int
Returns the number of articulation points.

articulation_points

```
articulation_points()
```

Finds the articulation points (cut points) of a graph.

Returns

list
Returns a list of articulation points in the graph.

assortativity_degree

```
assortativity_degree(source_direction='out', target_direction='in',
edge_weight=None)
```

Computes a graph's degree assortativity.

Parameters

source_direction : {'in', 'out'}
Specifies the degree type (in-degree or out-degree) for the source node in a directed graph.

target_direction : {'in', 'out'}
Specifies the degree type (in-degree or out-degree) for the target node in a directed graph.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted assortativity degree is computed.

Returns

float
Returns the degree assortativity.

assortativity_nominal

```
assortativity_nominal(node_attr)
```

Computes a graph's nominal assortativity.

Parameters

node_attr : str
Specifies the name of the nominal node attribute to use.

Returns

float
Returns the nominal assortativity.

assortativity_numeric

```
assortativity_numeric(node_attr)
```

Computes a graph's numeric assortativity.

Parameters

node_attr : str
Specifies the name of the numeric node attribute to use.

Returns

float
Returns the numeric assortativity.

average_shortest_path

```
average_shortest_path(edge_weight=None)
```

Computes a graph's average shortest path.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted average shortest path is computed.

Returns

float
Returns the average shortest path.

biconcomp_distribution_edges

```
biconcomp_distribution_edges(bins=None)
```

Computes the distribution of the number of edges in each biconnected components by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is int (an integer), it specifies the number of equal-width bins. When *bins* is a list, it specifies bin ranges and should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is str (a string), it specifies the method to use for calculating the bin ranges. For example, 'stone','auto','doane','fd','rice','scott','sqrt','sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of biconnected components whose edge count is within the range of the bin.

biconcomp_distribution_nodes

```
biconcomp_distribution_nodes(bins=None)
```

Computes the distribution of the number of nodes in each biconnected components by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is int (an integer), it specifies the number of equal-width bins. When *bins* is a list, it specifies bin ranges and should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is str (a string), it specifies the method to use for calculating the bin ranges. For example, 'stone','auto','doane','fd','rice','scott','sqrt','sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of biconnected components whose node count is within the range of the bin.

biconcomp_size_edges

```
biconcomp_size_edges()
```

Computes the number of edges within each biconnected component.

Returns

dict
Returns a dictionary in which the keys are biconnected component IDs and the values are the number of edges within each component.

biconcomp_size_nodes

```
biconcomp_size_nodes()
```

Computes the number of nodes within each biconnected component.

Returns

dict
Returns a dictionary in which the keys are biconnected component IDs and the values are the number of nodes within each component.

biconnected_component_count

```
biconnected_component_count()
```

Computes a graph's biconnected component count.

Returns

int
Returns the number of biconnected components.

biconnected_components

```
biconnected_components(order_by=None, *, data=True, ascending=False)
```

Computes the graph's biconnected components for an undirected graph.

Parameters

order_by : 'node' or 'edge' or None, optional
Specifies the parameter that is used to sort the components by the number of nodes or edges within each component. The default is None, which means that the components are not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Returns the result as a generator of biconnected component subgraphs. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

block_cut_tree

```
block_cut_tree(*, return_nodes_map=False)
```

Computes the block-cut tree of a graph.

Parameters

return_nodes_map : bool, optional
Specifies whether or not to also compute the block-cut tree node map. The default is False.

Returns

SAS graph object
Returns a graph that represents the block-cut tree. When *return_nodes_map = True*, the output graph has the *nodes_map* attribute, which contains a dictionary that maps its nodes to a list of nodes in the original graph. Note: If a block is empty (that is, the corresponding biconnected component contains only articulation points), that block does not appear as a key in the *nodes_map* dictionary.

centrality_betweenness

```
centrality_betweenness(edge_weight=None, sample_percent=100, *,
inplace=False, data=True)
```

Computes weighted or unweighted betweenness centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that unweighted betweenness centrality is computed.

sample_percent : int, optional
Specifies the percentage of source nodes to sample for the approximate betweenness calculation. This parameter value should be in the range (0, 100]. The default is 100.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the node and edge attributes *between_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_closeness

```
centrality_closeness(edge_weight=None, nopath='diameter', *, inplace=False,
data=True)
```

Computes weighted or unweighted closeness centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted closeness centrality is computed.

nopath : {'diameter', 'harmonic', or 'zero'}, optional
Specifies a method to use for accounting for the shortest path distance between two nodes when a path does not exist. The valid values are as follows: - 'diameter' uses the graph diameter (plus one) as the shortest path distance between disconnected nodes. - 'harmonic' uses the harmonic formula for calculating closeness centrality. - 'zero' uses zero as the shortest path distance between disconnected nodes.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the closeness centrality node attributes *centr_close_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_degree

```
centrality_degree(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted degree centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted degree centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the degree centrality node attributes *centr_degree_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_eigenvector

```
centrality_eigenvector(edge_weight=None, maxiters=10000, *, inplace=False,
data=True)
```

Computes weighted or unweighted eigenvector centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted eigenvector centrality is computed.

maxiters : int, optional
Specifies the maximum number of iterations in order to limit the amount of computation time that is spent when convergence is slow. The default is 10,000.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the degree centrality node attributes *centr_eigen_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_hub_authority

```
centrality_hub_authority(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted hub and authority centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted hub and authority centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the hub and authority centrality node attributes *centr_hub_** and *centr_auth_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_influence

```
centrality_influence(edge_weight=None, *, inplace=False, data=True)
```

Computes weighted or unweighted influence centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted influence centrality is computed.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the first- and second-order influence centrality node attributes *centr_influence1_** and *centr_influence2_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

centrality_pagerank

```
centrality_pagerank(edge_weight=None, alpha=0.85, tolerance=1e-09, *,
inplace=False, data=True)
```

Computes weighted or unweighted PageRank centrality and stores the results as attributes on the graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that unweighted PageRank centrality is computed.

alpha : float, optional
Specifies the damping parameter for the PageRank centrality (or algorithm). The default is 0.85.

tolerance : float, optional
Specifies the tolerance parameter for the PageRank centrality (or algorithm). The default is 1E-9.

inplace : bool, optional
Specifies whether to add the centrality values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that contains the PageRank centrality node attributes *centr_pagerank_**. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

clear

```
clear(*args, **kwargs)
```

Removes all nodes and edges from the graph.

clear_edges

```
clear_edges(*args, **kwargs)
```

Removes all edges from the graph but retains nodes.

concomp_distribution_edges

```
concomp_distribution_edges(bins=None, *, strongly=True)
```

Computes the distribution of the number of edges in each connected component by using *numpy.histogram*.

Parameters

bins : int or list or str, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is an integer (int), it represents the number of equal-width bins. When *bins* is a list, it specifies bin ranges and it should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)*

(inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is a string (str), it specifies the method to be used for calculating the bin ranges. For example, 'stone', 'auto', 'doane', 'fd', 'rice', 'scott', 'sqrt', 'sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of components whose edge count within the range of the bin.

concomp_distribution_nodes

```
concomp_distribution_nodes(bins=None, *, strongly=True)
```

Computes the distribution of the number of nodes in each connected component by using *numpy.histogram*.

Parameters

bins : int or list or str or None, optional
Specifies the bins for the distribution calculation. The default is None. When *bins* is None, bins are defined as the list of unique component sizes. When *bins* is an integer (int), it represents the number of equal-width bins. When *bins* is a list, it specifies bin ranges and it should be monotonically increasing. For example, a value of [a1, a2, a3, a4] results in the following bin ranges: *[a1, a2)* (inclusive of *a1* but exclusive of *a2*), *[a2, a3)*, and *[a3, a4]*. When *bins* is a string (str), it specifies the method to use for calculating the bin ranges. For example, 'stone', 'auto', 'doane', 'fd', 'rice', 'scott', 'sqrt', 'sturges'. See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for more information about the *bins* parameter.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary where the key is the bin's lower-bound value and the value is the number of components whose node count is within the range of the bin.

concomp_size_edges

```
concomp_size_edges(*, strongly=True)
```

Computes the number of edges within each connected component.

Parameters

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary in which the keys are connected component IDs and the values are the number of edges within each component.

concomp_size_nodes

```
concomp_size_nodes(*, strongly=True)
```

Computes the number of nodes within each connected component.

Parameters

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

Returns

dict
Returns a dictionary in which the keys are connected component IDs and the values are the number of nodes within each component.

connected_component_count

```
connected_component_count()
```

Computes a graph's connected component count.

Returns

int
Returns the number of connected components.

connected_components

```
connected_components(order_by=None, *, strongly=True, data=True,
ascending=False)
```

Computes the connected components of a graph.

Parameters

order_by : 'node' or 'edge' or None, optional
Specifies whether to sort the components by the number of nodes or edges within each component. The default is None, which means that the components are not sorted.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one connected component. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

copy

```
copy(immutable=None, *, data=True)
```

Returns a copy of the input graph.

Parameters

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, meaning that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns an independent copy of the input.

core_decomposition

```
core_decomposition(maxtime=None, *, data=True)
```

Computes a core decomposition of the graph.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time for core decomposition to spend. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one k-core in increasing core number. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

cycle_count

```
cycle_count(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Counts the number of cycles in a graph.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Returns

int
Returns a count of cycles in the graph.

degree

```
degree(n=None, edge_weight=None)
```

Returns the degree of nodes.

Parameters

n : single node or container of nodes or None, optional
Specifies which nodes to include. The default is None, which means that the degree of every node is returned.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None.

Returns

int or dictionary
Returns the degree of specified node(s).

density

```
density()
```

Computes the density of the graph.

Returns

float
Returns the density of the graph.

diameter

```
diameter(edge_weight=None)
```

Computes a graph's diameter.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted diameter is computed.

Returns

int or float
Returns the diameter.

drop_edge_attributes

```
drop_edge_attributes(attributes_subset=None, edges_subset=None,
immutable=None, *, inplace=False)
```

Removes edge attributes if they exist.

Parameters

attributes_subset : str or list, optional
Specifies the edge attribute or a list of the edge attributes to remove; for example, 'weight' or ['weight','color']. By default, all edge attributes are removed.

edges_subset : list or iterable of edges, optional
Specifies the edges to remove attributes from; for example, [(1,2),(2,3)]. By default the attributes for all edges are removed.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by removing the specified edge attributes. If the *inplace* parameter value is False, a copy of the input graph with the modified edge attributes is returned.

drop_node_attributes

```
drop_node_attributes(attributes_subset=None, nodes_subset=None,
immutable=None, *, inplace=False)
```

Removes node attributes if they exist.

Parameters

attributes_subset : str or list, optional
Specifies the node attribute or list of node attributes to remove; for example, ['weight','color']. By default, all node attributes are removed.

nodes_subset : list or iterable of nodes, optional
Specifies the nodes to drop attributes from; for example, [1,2,3] or (1,2,3) or [1] or (1,). By default, the attributes for all nodes are removed.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by removing the specified node attributes. If the *inplace* parameter value is False, a copy of the input graph with the modified node attributes is returned.

enumerate_cycle_edges

```
enumerate_cycle_edges(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Enumerates the cycles in a graph and returns the edges of the cycles.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Yields

generator
Yields a generator of cycles, each of which is given as a list of the cycles' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_cycle_nodes

```
enumerate_cycle_nodes(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None)
```

Enumerates the cycles in a graph and returns the nodes of the cycles.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

Yields

generator
Yields a generator of cycles, each of which is given as a list of the cycles' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_cycles

```
enumerate_cycles(maxcycles='ALL', minlength=1, maxlength=None,
node_weight=None, edge_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, source=None, *,
data=True)
```

Enumerates the cycles in a graph and returns the cycle graph objects.

Parameters

maxcycles : int or 'ALL', optional
Specifies the maximum number of cycles to count. The default is 'ALL'.

minlength : int, optional
Specifies the minimum number of edges in cycles that are counted. The default is 1.

maxlength : int, optional
Specifies the maximum number of edges in cycles that are counted. The default is None.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None. When the value is None, a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None. When the value is None, a weight of 1 is assumed for all edges.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a cycle. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a cycle. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a cycle. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a cycle. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds to spend on enumeration. The default is None.

source : node or None, optional
Specifies the source node for cycle calculations. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one cycle of the same type as the input graph. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_path_edges

```
enumerate_path_edges(source=None, sink=None, minlength=1,
maxlength=None, edge_weight=None, node_weight=None, minedgewt=None,
maxedgewt=None, minnodewt=None, maxnodewt=None, maxtime=None)
```

Enumerates the paths in a graph and returns the edges of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

Yields

generator
Yields a generator of Path objects, each of which is given as a list of the paths' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_path_nodes

```
enumerate_path_nodes(source=None, sink=None, minlength=1,
maxlength=None, edge_weight=None, node_weight=None, minedgewt=None,
maxedgewt=None, minnodewt=None, maxnodewt=None, maxtime=None)
```

Enumerates the paths in a graph and returns the nodes of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

Yields

generator
Yields a generator of Path objects, each of which is given as a list of the paths' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_paths

```
enumerate_paths(source=None, sink=None, minlength=1, maxlength=None,
edge_weight=None, node_weight=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, *, data=True)
```

Enumerates the paths in a graph and returns the Path graph objects.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

minlength : int, optional
Specifies the minimum number of edges in paths that are returned. The default is 1.

maxlength : int or None, optional
Specifies the maximum number of edges in paths that are returned. The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

minedgewt : float or None, optional
Specifies the minimum edge weight of edges in paths that are returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum edge weight of edges in paths that are returned. The default is None.

minnodewt : float or None, optional
Specifies the minimum node weight of nodes in paths that are returned. The default is None.

maxnodewt : float or None, optional
Specifies the maximum node weight of nodes in paths that are returned. The default is None.

maxtime : float or None, optional
Specifies the maximum amount of time for enumeration to spend. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one Path object. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_sequence_shortest_paths

```
enumerate_sequence_shortest_paths(sequence, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None, *, data=True)
```

Enumerates the shortest paths in a graph that visit the indicated sequence nodes in order.

Parameters

sequence : iterable of nodes
Specifies the order in which to visit the required nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Path objects; each object represents a subpath between a sequence of nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_path_edges

```
enumerate_shortest_path_edges(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None)
```

Enumerates the shortest paths in a graph and returns the edges of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

Yields

generator
Yields a generator of paths; each path is given as a list of the paths' edges. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_path_nodes

```
enumerate_shortest_path_nodes(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None)
```

Enumerates the shortest paths in a graph and returns the nodes of the paths.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

Yields

generator
Yields a generator of paths; each path is given as a list of the paths' nodes. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

enumerate_shortest_paths

```
enumerate_shortest_paths(source=None, sink=None, pathsperpair=1,
edge_weight=None, minedgewt=None, maxedgewt=None, maxabsobjgap=None,
maxrelobjgap=None, *, data=True)
```

Enumerates the shortest paths in a graph and returns the Path graph objects.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as source nodes.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) for shortest path enumeration. The default is None, which means that all nodes within the graph are used as sink nodes.

pathsperpair : int, optional
Specifies the maximum number of shortest paths to return for each source-sink pair. The default is 1.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

maxabsobjgap : nonnegative float or None, optional
Specifies the maximum weight by which a returned path can exceed the optimum shortest path weight. The default is None.

maxrelobjgap : nonnegative float or None, optional
Specifies the maximum factor by which a returned path can exceed the optimum shortest path weight. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Path objects; each object represents one shortest path. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

get_meta_graph

```
get_meta_graph(node_type, edge_type=None, immutable=None)
```

Constructs a metagraph from the input graph.

Parameters

node_type : str
Specifies the node attribute to consider in constructing the metagraph.

edge_type : str, optional
Specifies the edge attribute to consider in constructing the metagraph. By default, all connections between nodes are considered the same.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

Returns

SAS graph object
Returns the metagraph, which has an edge attribute count and determines the number of unique connections between pairs of nodes.

get_nlargest_biconcomps

```
get_nlargest_biconcomps(n, order_by='node', *, data=True, ascending=False)
```

Returns a list of up to *n* largest biconnected components.

Parameters

n : int
Specifies the number of graphs to return.

order_by : 'node' or 'edge' or None
Specifies whether to sort the biconnected components by the number of nodes or edges within those components. The default is None, which means that the biconnected components are not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. When the value is True, the smallest biconnected components are returned instead. The default is False.

Returns

list
Returns a list of n largest or smallest biconnected components of a graph.

get_nlargest_concomps

```
get_nlargest_concomps(n, order_by='node', *, strongly=True, data=True,
ascending=False)
```

Returns a list of up to *n* largest connected (with highest number of nodes) components of a graph.

Parameters

n : int
Specifies the number of graphs to return.

order_by : 'node' or 'edge'
Specifies whether to sort the component by number of nodes or edges within that component. The default is None, which means that the result is not sorted.

strongly : bool, optional
Specifies whether or not the connected components must be strongly connected. When the value is False, only weakly connected components are found. This parameter is supported only for directed graphs. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Returns

list
Returns a list of the n largest connected components of a graph.

has_edge_attribute

```
has_edge_attribute(name, *, all_edges=False)
```

Checks the edges of the input graph for the specified edge attribute.

Parameters

name : str
Specifies the name of the edge attribute.

all_edges : bool, optional
Specifies whether or not to check for the existence of the edge attribute in all or any of the edges. The default is False.

Returns

bool
If the all_edges parameter value is True, the value True is returned if that edge attribute exists in all edges; otherwise the value False is returned. If the all_edges parameter value is False, the value True is returned if at least one edge in the graph has that specified edge attribute.

has_node_attribute

```
has_node_attribute(name, *, all_nodes=False)
```

Checks the nodes of the input graph for the specified node attribute.

Parameters

name : str
Specifies the name of the node attribute.

all_nodes : bool, optional
Specifies whether or not to check for the existence of the node attribute in all or any of the nodes. The default is False.

Returns

bool
If the all_nodes parameter value is True, the value True is returned if that node attribute exists in all node; otherwise the value False is returned. If the all_nodes parameter value is False, the value True is returned if at least one node in the graph has that specified node attribute.

in_degree

```
in_degree(n=None, edge_weight=None)
```

Returns the in-degree of nodes.

Parameters

n : single node or container of nodes or None, optional
Specifies which nodes to include. The default is None, which means that the in-degree of all nodes is returned.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None.

Returns

numeric or dictionary
Returns the in-degree of the node if a single node is passed, or returns a dictionary with nodes as keys and their in-degrees as values if None or a list of nodes is passed.

intersection

```
intersection(*args, **kwargs)
```

Returns the intersection of input graphs by finding the intersections of their nodes and edges.

The node attributes and edge attributes are aggregated using the rules that are defined by the *nodes_attrs_agg* and *edges_attrs_agg* parameters.

Parameters

rhs : SAS graph object
Specifies the other graph, of the same type as *self*, with which to intersect.

nodes_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate node attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps node attributes to functions. The default is None.

edges_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate edge attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps edge attributes to functions. The default is None.

Returns

SAS graph object
Returns a graph that is constructed by the intersection of the two graphs.

is_biconnected

```
is_biconnected()
```

Specifies whether or not the input graph is biconnected.

Returns

bool
Indicates whether the input graph is biconnected (True) or not (False).

is_bipartite

```
is_bipartite()
```

Determines whether the input graph is bipartite.

Returns

bool
Indicates whether the input graph is bipartite.

is_connected

```
is_connected()
```

Specifies whether or not the input graph is connected.

Returns

bool
Indicates whether the input graph is connected (True) or not (False).

is_cycle

```
is_cycle()
```

Determines whether the input graph is a simple cycle.

Returns

bool
Indicates whether the input graph is a simple cycle (True) or not (False).

is_dag

```
is_dag()
```

Determines whether the input graph is a directed acyclic graph (DAG).

Returns

bool
Indicates whether the input graph is a directed acyclic graph (DAG).

is_directed

```
is_directed()
```

Specifies whether or not the graph is directed.

Returns

bool
Indicates whether or not the graph is directed.

is_hamiltonian

```
is_hamiltonian(maxtime=None)
```

Determines whether the graph is Hamiltonian (has a Hamiltonian cycle) or not.

Parameters

maxtime : float or None, optional
Specifies the maximum amount of time to determine whether or not the graph is Hamiltonian. The default is None.

Returns

bool or None
Indicates whether or not the input graph is determined to be Hamiltonian. Returns None if no determination is made.

is_immutable

```
is_immutable()
```

Specifies whether or not the graph is immutable.

Returns

bool
Indicates whether or not the graph is immutable.

is_multigraph

```
is_multigraph()
```

Specifies whether or not the graph allows multiedges.

Returns

bool
Indicates whether or not the graph allows multiedges.

is_path

```
is_path()
```

Determines whether the input graph is a simple path.

Returns

bool
Indicates whether the input graph is a simple path (True) or not (False).

is_strongly_connected

```
is_strongly_connected()
```

Specifies whether the input graph is strongly connected.

Returns

bool
Indicates whether the input graph is strongly connected (True) or not (False).

is_weakly_connected

```
is_weakly_connected()
```

Specifies whether the input graph is weakly connected.

Returns

bool
Indicates whether the input graph is weakly connected (True) or not (False).

label_propagation

```
label_propagation(edge_weight=None, maxiters=100, tolerance=0.05, *,
data=True)
```

Computes the communities of the graph by using the label propagation algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of label changes for all nodes in the graph is less than the value. The default is 0.05.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one community. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

label_propagation_nodes

```
label_propagation_nodes(edge_weight=None, maxiters=100, tolerance=0.05, *,
data=True, inplace=False)
```

Computes community assignments for all nodes in the graph by using the label propagation algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of label changes for all nodes in the graph is less than the value. The default is 0.05.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

inplace : bool, optional
Specifies whether to add the community values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains the node attribute *community*, which denotes the community assignment for each node. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

leaf_node_count

```
leaf_node_count()
```

Computes a graph's leaf node count.

Returns

int
Returns the number of leaf nodes.

local_transitivity

```
local_transitivity(*, inplace=False)
```

Computes the local transitivity (clustering coefficient) and stores the results as node attributes on the graph.

Parameters

inplace : bool, optional
Specifies whether to add the transitivity values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains node attribute 'transitivity'. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

louvain

```
louvain(*, edge_weight=None, resolution=1, tolerance=0.001, maxiters=100,
data=True)
```

Computes the communities of the graph by using the Louvain algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

resolution : float, optional
Specifies the factor in the modularity equation that affects the expected number of links between two nodes. A higher value produces more communities. The default is 1.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of modularity gain between two consecutive iterations is less than the value. The default is 0.001.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one community. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

louvain_nodes

```
louvain_nodes(*, edge_weight=None, resolution=1, tolerance=0.001,
maxiters=100, data=True, inplace=False)
```

Computes community assignments for all nodes in the graph by using the Louvain algorithm.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

resolution : float, optional
Specifies the factor in the modularity equation that affects the expected number of links between two nodes. A higher value produces more communities. The default is 1.

tolerance : float, optional
Specifies the value for which the algorithm stops iterating when the fraction of modularity gain between two consecutive iterations is less than the value. The default is 0.001.

maxiters : int, optional
Specifies the number of iterations that the algorithm can run. The default is 100.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

inplace : bool, optional
Specifies whether to add the community values as attributes to the input graph directly or to create a copy of the graph. The default is False.

Returns

SAS graph object or None
Returns a graph that contains the node attribute *community*, which denotes the community assignment for each node. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

maxflow

```
maxflow(source, sink, capacity_attr='upper', *, inplace=False, data=True)
```

Calculates the maximum flow from a source node to a sink node in a graph.

Parameters

source : number or str
Specifies the source node for the maximum flow.

sink : number or str
Specifies the sink node for the maximum flow.

capacity_attr : str, optional
Specifies the name of the edge attribute that contains the capacity of the edge. The default is 'upper'.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute *flow*, which contains the flow values. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, the returned graph contains only the flow edges.

maximal_cliques

```
maximal_cliques(maxcliques='ALL', node_weight=None, edge_weight=None,
minsize=1, maxsize=None, minedgewt=None, maxedgewt=None,
minnodewt=None, maxnodewt=None, maxtime=None, order_by=None, *,
data=True, ascending=False)
```

Enumerates all maximal cliques of the graph.

Parameters

maxcliques : int or 'ALL', optional
Specifies the maximum number of enumerated cliques to return. The default is 'ALL'.

node_weight : str or None, optional
Specifies the node attribute that corresponds to the node weight. The default is None, which means that a weight of 1 is assumed for all nodes.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

minsize : int, optional
Specifies the minimum number of nodes in a clique. The default is 1.

maxsize : int or None, optional
Specifies the maximum number of nodes in a clique. The default is None.

minedgewt : float or None, optional
Specifies the minimum sum of edge weights in a clique. The default is None.

maxedgewt : float or None, optional
Specifies the maximum sum of edge weights in a clique. The default is None.

minnodewt : float or None, optional
Specifies the minimum sum of node weights in a clique. The default is None.

maxnodewt : float or None, optional
Specifies the maximum sum of node weights in a clique. The default is None.

maxtime : float or None, optional
Specifies the approximate maximum number of seconds for enumeration to spend. The default is None.

order_by : 'node' or None, optional
Specifies whether to sort the cliques by the number of nodes within each clique. The default is None, which means that the result is not sorted.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

ascending : bool, optional
Specifies whether to sort in ascending or descending order, when you specify the *order_by* parameter. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one maximal clique. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

min_cut

```
min_cut(edge_weight=None, *, inplace=False, data=True)
```

Calculates a minimum cut of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted minimum cut is calculated.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute 'mincut', which indicates the edges to include in the minimum cut set, and the node attribute 'mincut_partition', which indicates the node partition of the calculated minimum cut. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, a modified copy of the graph is returned.

min_s_t_cut

```
min_s_t_cut(source, sink, edge_weight=None, *, inplace=False, data=True)
```

Calculates a minimum s-t cut of a graph.

Parameters

source : number or str
Specifies the source node (s) for the minimum s-t cut.

sink : number or str
Specifies the sink node (t) for the minimum s-t cut.

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted minimum s-t cut is calculated.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute *s_t_cut*, which indicates the edges to include in the minimum s-t cut set; and the node attribute *s_t_cut_partition*, which indicates the node partition of the calculated minimum s-t cut. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, a modified copy of the graph is returned.

mincostflow

```
mincostflow(demand_lower_attr='lower', demand_upper_attr='upper',
capacity_lower_attr='lower', capacity_upper_attr='upper', cost_attr='weight',
maxtime=None, *, inplace=False, data=True)
```

Calculates the minimum-cost network flow for the graph.

Parameters

demand_lower_attr : str, optional
Specifies the name of the node attribute that contains the lower bound of the node supply. The default is 'lower'.

demand_upper_attr : str, optional
Specifies the name of the node attribute that contains the upper bound of the node supply. The default is 'upper'.

capacity_lower_attr : str, optional
Specifies the name of the edge attribute that contains the capacity lower bound of the edge. The default is 'lower'.

capacity_upper_attr : str, optional
Specifies the name of the edge attribute that contains the capacity upper bound of the edge. The default is 'upper'.

cost_attr : str, optional
Specifies the name of the edge attribute that contains the edge cost. The default is 'weight'.

maxtime : float or None, optional
Specifies the maximum amount of time to allow for computing the minimum-cost network flow. The default is None.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attributes *flow* and *reduced_cost*, which contain the flow and reduced cost values, respectively; and the node attribute *dual*, which contains the optimal dual value. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned.

minimum_spanning_tree

```
minimum_spanning_tree(edge_weight=None, source=None, *, inplace=False,
data=True)
```

Calculates a minimum spanning tree (MST) or forest of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

source : number or str or None
Specifies the source node for a directed MST. The default is None.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the edge attribute 'mst', which indicates the edges to include in the spanning tree or forest. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is True, the graph is modified in place and None is returned. When the *inplace* parameter value is False, the returned graph contains only the spanning tree or forest edges.

minimum_weight_full_matching

```
minimum_weight_full_matching(edge_weight=None, *, data=True)
```

Returns a minimum-weight full matching of the bipartite graph *self*.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns a directed graph whose edges identify the assignment mapping in the minimum-weight full matching solution. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

neighbors

```
neighbors(n)
```

Returns an iterator over all neighbors of node n.

Parameters

n : node
Specifies a node in the graph.

Returns

iterator
Returns an iterator over all neighbors of node n.

node_clique_numbers

```
node_clique_numbers(*, inplace=False, data=True)
```

Computes node clique numbers and stores the results as attributes on the graph.

Parameters

inplace : bool, optional
Specifies whether to add the clique number values as attributes to the input graph directly or to create a copy of the graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the value of the *inplace* parameter is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph or modifies the existing graph to contain the node attribute *clique_number*, which includes the clique number of each node. The graph g has the attribute g.solution_summary, which contains information about the results of the algorithm.

number_of_edges

```
number_of_edges(u=None, v=None)
```

Returns the number of edges in the graph.

When you specify *u* and *v*, the number of edges from *u* to *v* is returned. When you specify *u* but *v* is None, the number of connections from *u* to other nodes (out-degree of node *u*) is returned. When you specify *v* but *u* is None, the number of connections from *v* to other nodes (in-degree of node *v*) is returned.

Parameters

u : numeric or str or None, optional
Specifies the *from* node. The default is None.

v : numeric or str or None, optional
Specifies the *to* node. The default is None.

Returns

int
Returns the number of edges in the graph. When you specify *u* and *v*, the number of edges from *u* to *v* is returned. When you specify *u* but *v* is None, the number of connections from *u* to other nodes(out-degree of node *u*) is returned. When you specify *v* but *u* is None, the number of connections from *v* to other nodes(in-degree of node *v*) is returned.

number_of_nodes

```
number_of_nodes()
```

Returns the number of nodes in the graph.

Returns

int
Returns the number of nodes in the graph.

out_degree

```
out_degree(n=None, edge_weight=None)
```

Returns the out-degree of nodes.

Parameters

n : single node or container of nodes or None, optional
Specifies which nodes to include. The default is None, which means that the out-degree of all nodes is returned.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None.

Returns

numeric or dictionary
Returns the out-degree of the node if a single node is passed, or returns a dictionary with nodes as keys and their out-degrees as values if None or a list of nodes is passed.

path_length

```
path_length(edge_weight=None)
```

Returns the total length of the path (edges).

Parameters

edge_weight : str, optional
Specifies the name of an edge attribute to use as the weight of edges. By default, edges are considered to be unweighted (weight=1).

Returns

number
Returns the total length of the path edges.

predecessors

```
predecessors(n)
```

Returns an iterator over all predecessor nodes of n.

Parameters

n : node
Specifies a node in the graph.

Returns

iterator
Returns an iterator over predecessors of node n.

projected_graph

```
projected_graph(nodes, neighbors=None, directed_method=None,
_measure=None, *, multigraph=False, data=True)
```

Computes a projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in self other than *nodes*.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used. When the *directed_method* parameter is specified for an undirected graph, a ValueError is raised.

multigraph : bool, optional
Specifies whether or not to return a multigraph. When the value is True, the algorithm returns a multigraph in which the multiple edges represent multiple shared neighbors. The edge attribute 'neighbor' stores the label of the shared neighbor. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

projected_graph_adamic_adar

```
projected_graph_adamic_adar(nodes, neighbors=None, directed_method=None,
*, data=True)
```

Computes an Adamic-Adar projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than *nodes*.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'adamic_adar' edge attribute.

projected_graph_common_neighbors

```
projected_graph_common_neighbors(nodes, neighbors=None,
directed_method=None, *, data=True)
```

Computes a common neighbors projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the converging method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'common_neighbors' edge attribute.

projected_graph_cosine

```
projected_graph_cosine(nodes, edge_weight=None, neighbors=None,
directed_method=None, *, data=True)
```

Computes a cosine projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'cosine' edge attribute.

projected_graph_dice

```
projected_graph_dice(nodes, neighbors=None, directed_method=None, *,
data=True)
```

Computes a Sørensen-Dice projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the subset of nodes that are specified by the *neighbors* parameter. The strength of association in the calculated projection is represented by the 'dice' edge attribute.

projected_graph_jaccard

```
projected_graph_jaccard(nodes, neighbors=None, directed_method=None, *,
data=True)
```

Computes a Jaccard projected graph.

Parameters

nodes : list or iterable
Specifies the subset of nodes to project onto.

neighbors : list or iterable, optional
Specifies the subset of nodes to project through. By default, this subset consists of all nodes in g other than those that are specified by the *nodes* parameter.

directed_method : {'converging', 'diverging', 'transitive'}, optional
Specifies a method to use for projecting directed graphs. By default, the 'converging' method is used.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained.

Returns

SAS graph object
Returns a graph representation of the node pairs with common neighbors among the *neighbors* subset of nodes. The strength of association in the calculated projection is represented by the 'jaccard' edge attribute.

query

```
query(q, expr=None, _node_attr_to_ignore=None, _edge_attr_to_ignore=None, *,
induced=False, break_symmetry=False, _generate_output=True,
_orbits_only=False, **kwargs)
```

Queries for the subgraph isomorphisms of q within g.

Parameters

q : graph
Specifies the query graph.

expr : _Expression or None, optional
Specifies additional logic to filter the query results. The default is None.

induced : bool, optional
Specifies whether or not to return only node-induced matches. The default is False.

break_symmetry : bool, optional
Specifies whether or not to break symmetry and eliminate automorphic relationships between matches. The default is False.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one isomorphic mapping from *q* to *self*. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

reach_graph

```
reach_graph(source, max_reach=1, *, data=True)
```

Computes the reach graph for the source or set of sources.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to compute the reach network.

max_reach : int, optional
Specifies the number of steps (or hops) to traverse from the source nodes. The default is 1.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns a graph that represents the reach network. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

reach_graph_per_source

```
reach_graph_per_source(source, max_reach=1, *, data=True)
```

Computes a separate reach graph for each given source node.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to compute the reach network.

max_reach : int, optional
Specifies the number of steps (or hops) to traverse from the source nodes. The default is 1.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of subgraphs; each subgraph represents one reach network. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

remove_edge

```
remove_edge(*args, **kwargs)
```

Removes a single edge from the graph.

Parameters

u : numeric or str
Specifies the *from* node.

v : numeric or str
Specifies the *to* node.

Examples

```
G.remove_edge(1,2)
```

remove_edges_from

```
remove_edges_from(*args, **kwargs)
```

Removes specified edges from the graph.

Parameters

edges : an iterable container of edges
Specifies the edges to remove from the graph.

remove_isolated_nodes

```
remove_isolated_nodes(*args, **kwargs)
```

Removes isolated nodes from a graph.

Parameters

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the input graph. The default is False.

Returns

SAS graph object or None
Returns the resulting graph after isolated nodes are removed, or returns nothing if the *inplace* parameter value is True.

remove_node

```
remove_node(*args, **kwargs)
```

Removes a single node from the graph.

Parameters

node : numeric or str
Specifies the node to remove from the graph.

remove_nodes_from

```
remove_nodes_from(*args, **kwargs)
```

Removes specified nodes from the graph.

If a specified node does not exist in the graph, it is silently ignored.

Parameters

nodes : an iterable container of nodes
Specifies the nodes to remove from the graph.

reverse

```
reverse(*, data=True)
```

Returns a copy of a directed graph with reversed edges.

Parameters

data : bool, optional
Specifies whether or not to retain attributes. The default is True, meaning that the node and edge attributes of the graph are retained.

Returns

directed graph object
Returns a directed graph that holds the reversed edges.

set_edge_attributes

```
set_edge_attributes(values, name=None, immutable=None, *, inplace=False)
```

Sets edge attributes by using a given value or dictionary of values.

Parameters

values : scalar or dict-like or Pandas DataFrame
Specifies what the edge attribute should be set to. When this parameter is set to a Pandas DataFrame, it is expected to have from, to, and edge_key (only for multigraphs) columns, and the other columns are treated as edge attributes to add. When *values* is a dictionary, the keys are edges and the values are either edge attribute values or dictionaries of attribute name-value pairs. For multigraphs, the edge tuples must be of the form (u, v, key). For simple graphs, the keys must be tuples of the form (u, v). When this parameter is not set to a dictionary, it is treated as a single attribute value that is then applied to every edge in *self*. The attribute name is specified by the *name* parameter.

name : str or None, optional
Specifies the name of the edge attribute to set if the *values* parameter is set to a scalar. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is False, a copy of the input graph with the new edge attributes is returned. If the *inplace* parameter value is True, the graph is modified by setting the specified edge attributes.

set_node_attributes

```
set_node_attributes(values, name=None, immutable=None, *, inplace=False)
```

Sets node attributes from a given value or dictionary of values.

Parameters

values : scalar or dict-like or Pandas DataFrame
Specifies what the node attribute should be set to. When *values* is not a dictionary, it is treated as a single attribute value that is then applied to every node in *self*. This means that if you provide a mutable object, such as a list, updates to that object are reflected in the node attribute for every node. The attribute name is specified by the *name* parameter. When *values* is a dictionary, the keys are nodes and the values are either node attribute values or dictionaries of attribute name-value pairs.

name : str or None, optional
Specifies the name of the node attribute to set if the *values* parameter is set to a scalar. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not the graph is modified in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by setting the specified node attributes. If the *inplace* parameter value is False, a copy of the input graph with the new node attributes is returned.

shortest_path_distances

```
shortest_path_distances(source=None, sink=None, edge_weight=None,
minedgewt=None, maxedgewt=None)
```

Calculates the shortest path distances in a graph.

Parameters

source : node or iterable of nodes or None, optional
Specifies the source node(s) and restricts enumeration to the shortest paths that contain the specified source node(s). The default is None.

sink : node or iterable of nodes or None, optional
Specifies the sink node(s) and restricts enumeration to the shortest paths that contain the specified sink node(s). The default is None.

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that the unweighted shortest path is computed.

minedgewt : float or None, optional
Specifies the minimum total edge weight for each shortest path that is returned. The default is None.

maxedgewt : float or None, optional
Specifies the maximum total edge weight for each shortest path that is returned. The default is None.

Returns

dict
Returns a dictionary where each key is a (source, sink) tuple and its value is the corresponding shortest source-sink path distance.

similarity_adamic_adar

```
similarity_adamic_adar(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Adamic-Adar node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_common_neighbors

```
similarity_common_neighbors(source, sink=None, top_k=None, bottom_k=None,
*, exclude_source=False, exclude_existing_neighbors=False)
```

Computes common neighbors node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_cosine

```
similarity_cosine(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes cosine node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_dice

```
similarity_dice(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Sørensen-Dice node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

similarity_graph

```
similarity_graph(measure, min_score=2.220446049250313e-16, max_score=inf,
*, exclude_source=False, exclude_existing_neighbors=False, data=True,
**kwargs)
```

Computes a node similarity graph.

Parameters

measure : str, one of {'adamic_adar', 'common_neighbors', 'cosine', 'dice', 'jaccard'}
Specifies the metric to be used to compute the node similarity.

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the edge attribute that corresponds to the measure name. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_adamic_adar

```
similarity_graph_adamic_adar(min_score=2.220446049250313e-16,
max_score=inf, *, exclude_source=False, exclude_existing_neighbors=False,
data=True)
```

Computes an Adamic-Adar node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'adamic_adar' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_common_neighbors

```
similarity_graph_common_neighbors(min_score=2.220446049250313e-16,
max_score=inf, *, exclude_source=False, exclude_existing_neighbors=False,
data=True)
```

Computes a common neighbors node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'common_neighbors' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_cosine

```
similarity_graph_cosine(edge_weight=None,
min_score=2.220446049250313e-16, max_score=inf, *, exclude_source=False,
exclude_existing_neighbors=False, data=True)
```

Computes a cosine node similarity graph.

Parameters

edge_weight : str or None, optional
Specifies the edge attribute that corresponds to the edge weight. The default is None, which means that a weight of 1 is assumed for all edges.

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'cosine' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_dice

```
similarity_graph_dice(min_score=2.220446049250313e-16, max_score=inf, *,
exclude_source=False, exclude_existing_neighbors=False, data=True)
```

Computes a Sørensen-Dice node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the *dice* edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_graph_jaccard

```
similarity_graph_jaccard(min_score=2.220446049250313e-16, max_score=inf, *,
exclude_source=False, exclude_existing_neighbors=False, data=True)
```

Computes a Jaccard node similarity graph.

Parameters

min_score : float, optional
Specifies the minimum similarity score for edges in the output graph. The default is sys.float_info.epsilon.

max_score : float, optional
Specifies the maximum similarity score for edges in the output graph. The default is np.inf.

exclude_source : bool, optional
Specifies whether or not to exclude self-loops from the returned graph. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned graph. The default is False.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node attributes of the graph are retained. Note that the edge attributes are not carried over to the output graph.

Returns

MultiGraph
Returns a MultiGraph representation of the node pairs whose similarity scores are in the range of *min_score* to *max_score*. The calculated similarity is represented by the 'jaccard' edge attribute. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

similarity_jaccard

```
similarity_jaccard(source, sink=None, top_k=None, bottom_k=None, *,
exclude_source=False, exclude_existing_neighbors=False)
```

Computes Jaccard node similarity.

Parameters

source : node or iterable of nodes
Specifies one or more source nodes from which to calculate similarity.

sink : node or iterable of nodes, optional
Specifies one or more source nodes from which to calculate similarity. By default, all nodes are sinks.

top_k : int, optional
Specifies the maximum number of highest-similarity source-sink pairs to return.

bottom_k : int, optional
Specifies the maximum number of lowest-similarity source-sink pairs to return.

exclude_source : bool, optional
Specifies whether or not to exclude the source-source pairs from the returned results. The default is False.

exclude_existing_neighbors : bool, optional
Specifies whether or not to exclude source-sink pairs that are joined by an existing link from the returned results. The default is False.

Returns

OrderedDict
Returns an ordered dictionary, sorted in decreasing order of similarity. The key is the (source, sink) tuple, and the value is the computed similarity score.

singleton_node_count

```
singleton_node_count()
```

Computes a graph's singleton node count.

Returns

int
Returns the number of singleton nodes.

successors

```
successors(n)
```

Returns an iterator over all successor nodes of n.

Parameters

n : node
Specifies a node in the graph.

Returns

iterator
Returns an iterator over successors of node n.

to_directed

```
to_directed()
```

Returns a directed copy of the input graph.

Returns

SAS graph object
Returns a directed copy of the input graph.

to_immutable

```
to_immutable()
```

Creates an immutable graph from the input graph.

Returns

SAS graph object
Returns an immutable SAS graph object.

to_mutable

```
to_mutable()
```

Creates a mutable graph from the input graph.

Returns

SAS graph object
Returns a mutable SAS graph object.

to_pandas_edges

```
to_pandas_edges(*, data=True, _for_conversion_to_immutable=False)
```

Returns the graph edges as a Pandas DataFrame.

Parameters

data : bool or list, optional
Specifies whether or not to retain attributes. The default is True. When the value is True (or when a list is given), the Pandas DataFrame includes all the edge attributes (or all the listed attributes).

Returns

dataframe
Returns a Pandas DataFrame that includes the graph edges.

to_pandas_nodes

```
to_pandas_nodes(*, data=True, _for_conversion_to_immutable=False)
```

Returns the graph nodes as a Pandas DataFrame.

Parameters

data : bool or list, optional
Specifies whether or not to retain attributes. The default is True. When the value is True (or when a list is given), the Pandas DataFrame includes all the node attributes (or all the listed attributes).

Returns

dataframe
Returns a Pandas DataFrame that includes the graph nodes.

to_undirected

```
to_undirected()
```

Returns a directed copy of the input graph.

Returns

SAS graph object
Returns a directed copy of the input graph.

topological_ordering

```
topological_ordering()
```

Finds a topological ordering of the graph's nodes.

Yields

generator
Yields a generator of nodes in topological order. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

transitive_closure

```
transitive_closure()
```

Calculates the transitive closure of a graph.

Returns

SAS graph object
Returns the transitive closure of the input graph that is returned as a SAS MultiGraph or MultiDiGraph object. This graph has the *solution_summary* attribute, which contains information about the results of the algorithm.

triangle_count

```
triangle_count()
```

Computes a graph's triangle count.

Returns

int
Returns the number of triangles.

tsp_tour

```
tsp_tour(edge_weight=None, maxtime=None, minobjective=None, *,
inplace=False, use_milp=True, data=True, **options)
```

Calculates a traveling salesman problem (TSP) tour of a graph.

Parameters

edge_weight : str or None, optional
Specifies the numeric edge attribute that corresponds to the edge weight. The default is None, which means that an unweighted TSP is calculated.

maxtime : float or None, optional
Specifies the maximum amount of time for solving a TSP. The default is None.

minobjective : float, optional
Specifies a minimum value such that when a solution is found whose objective is less than or equal to this value, the algorithm stops.

inplace : bool, optional
Specifies whether to add the TSP order values as attributes to the input graph directly or to create a copy of the graph. The default is False.

use_milp : bool, optional
Specifies whether or not to use a mixed integer linear programming (MILP) solver to solve a TSP. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. When the value is True and the *inplace* parameter value is False (default values), the node and edge attributes of the graph are retained. This parameter has no effect when the *inplace* parameter is set to True.

Returns

SAS graph object or None
Returns a graph that has the *solution_summary* attribute, which contains information about the results of the algorithm. When the *inplace* parameter value is False, the returned graph is a Cycle that contains the calculated TSP tour edges. When the *inplace* parameter value is True, the graph is modified in place and None is returned. The modified graph contains the edge attributes *tsp* and *tsp_order* and the node attribute *tsp_order*.

union

```
union(*args, **kwargs)
```

Returns the union of input graphs by finding the unions of their nodes and edges.

The node attributes and edge attributes are aggregated using the rules that are defined by the *nodes_attrs_agg* and *edges_attrs_agg* parameters.

Parameters

rhs : SAS graph object
Specifies the other graph, of the same type as *self*, with which to find the union.

nodes_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate node attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps node attributes to functions. The default is None.

edges_attrs_agg : function or str or dict or None, optional
Specifies how to aggregate edge attributes. The value can be a function, such as *mean* or *np.sum*; a string in the list *{'first','last','mean'}*; or a dictionary that maps edge attributes to functions. The default is None.

Returns

SAS graph object
Returns a graph that is constructed by the union of the two graphs.

vrp

```
vrp(capacity, depot, node_demand='demand', edge_cost='weight', minroutes=1,
maxroutes=None, maxtime=None, minobjective=None, *, use_milp=True,
data=True, **options)
```

Solves the vehicle routing problem.

Parameters

capacity : float
Specifies the capacity of each vehicle.

depot : node
Specifies the depot node for the vehicle routing problem.

node_demand : str or dict or iterable
Specifies the quantity of goods to deliver to or pick up from a customer. The default is 'demand'.

edge_cost : str or dict or iterable
Specifies the cost of traversing each edge. The default is 'weight'.

minroutes : int, optional
Specifies the minimum number of routes that are allowed to service demand.

maxroutes : int, optional
Specifies the maximum number of routes that are allowed to service demand.

maxtime : float or None, optional
Specifies the maximum amount of time for solving the vehicle routing problem. The default is None.

minobjective : float, optional
Specifies a minimum value such that when a solution is found whose objective is less than or equal to this value, the algorithm stops.

use_milp : bool, optional
Specifies whether or not to use a mixed integer linear programming (MILP) solver to solve the vehicle routing problem. The default is True.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Cycle objects; each cycle represents one route. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

vrptw

```
vrptw(capacity, depot, node_demand='demand', edge_cost='weight',
travel_time='weight', timewindow_lower=None, timewindow_upper=None,
service_time=None, minroutes=1, maxroutes=None, maxtime=None, *,
data=True)
```

Solves the vehicle routing problem with time windows.

Parameters

capacity : float
Specifies the capacity of each vehicle.

depot : node
Specifies the depot node for the vehicle routing problem.

node_demand : str or dict or iterable
Specifies the quantity of goods to deliver to or pick up from a customer. The default is 'demand'.

edge_cost : str or dict or iterable
Specifies the cost of traversing each edge.

travel_time : str or dict or iterable
Specifies attributes or values that define the time that it takes to traverse each edge. The default is 'weight'.

timewindow_lower : str or dict or iterable or None, optional
Specifies attributes or values that define the lower time limits within which a delivery vehicle can arrive at the customer nodes. The default is None.

timewindow_upper : str or dict or iterable or None, optional
Specifies attributes or values that define the upper time limits within which a delivery vehicle can arrive at the customer nodes. The default is None.

service_time : str or dict or iterable or None, optional
Specifies attributes or values that define the service time of the nodes. The default is None.

minroutes : int, optional
Specifies the minimum number of routes that are allowed to service demand.

maxroutes : int, optional
Specifies the maximum number of routes that are allowed to service demand.

maxtime : float or None, optional
Specifies the maximum amount of time for solving the vehicle routing problem. The default is None.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Yields

generator
Yields a generator of Cycle objects; each cycle represents one route. The generator has the *solution_summary* attribute, which contains information about the results of the algorithm.

25

Built-In Graph Utilities

Functions

Function Name	Description
barbell_graph	Returns a barbell graph with two cliques of size n that are connected by a path of length path_len.
binomial_graph	Returns a random Erdős-Rényi graph of size n and probability p.
clique_graph	Returns a complete graph of size n.
complete_graph	Returns a complete graph of size n.
cycle_graph	Returns a cycle graph with n nodes and n edges.
erdos_renyi_graph	Returns a random Erdős-Rényi graph of size n and probability p.
grid_graph	Returns a two-dimensional grid graph of size num_rows x num_cols.
karate_club_graph	Returns the Zachary's Karate Club graph.
path_graph	Returns a path graph with n nodes and n-1 edges.
simple_graphlets	Generates a list of all graphlets within the specified size range.

barbell_graph

Returns a barbell graph with two cliques of size n that are connected by a path of length path_len.

API: barbell_graph

```
sasviya.network.utils.built_in_graphs.barbell_graph(n, path_len)
```

Only undirected barbell graphs are supported.

Parameters

n : int
Specifies the size of each clique.

path_len : int
Specifies the length of the path that connects two cliques.

Returns

SAS graph object
Returns a barbell graph with two cliques of size n and a path of length path_len.

binomial_graph

Returns a random Erdős-Rényi graph of size n and probability p.

API: binomial_graph

```
sasviya.network.utils.built_in_graphs.binomial_graph(n, p, seed=None, *,
directed=False)
```

Parameters

n : int
Specifies the number of nodes in the graph.

p : int
Specifies the probability that each edge is included.

seed : int, optional
Specifies the random seed. By default, the global default random number generator is used.

directed : bool, optional
Specifies whether or not the graph is directed. The default is False.

Returns

SAS graph object
Returns a random graph of size n and probability p.

clique_graph

Returns a complete graph of size n.

API: clique_graph

```
sasviya.network.utils.built_in_graphs.clique_graph(n, *, directed=False)
```

Parameters

n : int
Specifies the size of the graph.

directed : bool, optional
Specifies whether or not the graph is directed. The default is False.

Returns

SAS graph object
Returns a complete graph of size n.

complete_graph

Returns a complete graph of size n.

API: complete_graph

```
sasviya.network.utils.built_in_graphs.complete_graph(n, *, directed=False)
```

Parameters

n : int
Specifies the size of the graph.

directed : bool, optional
Specifies whether or not the graph is directed. The default is False.

Returns

SAS graph object
Returns a complete graph of size n.

cycle_graph

Returns a cycle graph with n nodes and n edges.

API: cycle_graph

```
sasviya.network.utils.built_in_graphs.cycle_graph(n, *, directed=False)
```

Parameters

n : int
Specifies the size of the graph.

directed : bool, optional
Specifies whether or not the graph is directed. The default is False.

Returns

SAS graph object
Returns a cycle graph of size n.

erdos_renyi_graph

Returns a random Erdős-Rényi graph of size n and probability p.

API: erdos_renyi_graph

```
sasviya.network.utils.built_in_graphs.erdos_renyi_graph(n, p, seed=None, *,
directed=False)
```

Parameters

n : int
Specifies the number of nodes in the graph.

p : int
Specifies the probability that each edge is included.

seed : int, optional
Specifies the random seed. By default, the global default random number generator is used.

directed : bool, optional
Specifies whether or not the graph is directed. The default is False.

Returns

SAS graph object
Returns a random graph of size n and probability p.

grid_graph

Returns a two-dimensional grid graph of size num_rows x num_cols.

API: grid_graph

```
sasviya.network.utils.built_in_graphs.grid_graph(num_rows, num_cols)
```

Only undirected grids are supported.

Parameters

num_rows : int
Specifies the number of rows of nodes in the grid.

num_cols : int
Specifies the number of columns of nodes in the grid.

Returns

SAS graph object
Returns a two-dimensional grid graph of size num_rows x num_cols.

karate_club_graph

Returns the Zachary's Karate Club graph.

API: karate_club_graph

```
sasviya.network.utils.built_in_graphs.karate_club_graph()
```

Returns

SAS graph object
Returns the Zachary's Karate Club graph.

path_graph

Returns a path graph with n nodes and n-1 edges.

API: path_graph

```
sasviya.network.utils.built_in_graphs.path_graph(n, *, directed=False)
```

Parameters

n : int
Specifies the size of the graph.

directed : bool, optional
Specifies whether or not the graph is directed. The default is False.

Returns

SAS graph object
Returns a path graph of size n.

simple_graphlets

Generates a list of all graphlets within the specified size range.

API: simple_graphlets

```
sasviya.network.utils.built_in_graphs.simple_graphlets(minsize=1, maxsize=7, *,
include_disconnected=False)
```

Parameters

minsize : int, optional
Specifies the number of nodes in the smallest graphs to return. The value must be at least 1 and at most 7. The default is 1.

maxsize : int, optional
Specifies the number of nodes in the largest graphs to return. The value must be at least 1 and at most 7. The default is 7.

include_disconnected : bool, optional
Specifies whether or not to include disconnected graphs. The default is False.

Yields

generator
Yields a generator of graphs; each graph object represents one graphlet.

26

Conversion Utilities

Functions

Function Name	Description
edge_attrs_to_torch_tensor	Returns edge attributes of the graph in the form of a torch tensor.
from_dgl	Converts a *DGL graph* instance to its corresponding SAS graph.
from_gds	Creates a SAS graph equivalent to the input Neo4j in-memory graph.
from_graphtool	Converts a graph-tool graph object to its equivalent SAS graph.
from_igraph	Converts an igraph graph to its equivalent SAS graph.
from_molecule	Converts a SMILES string or *rdkit* molecule to a SAS graph.
from_neo4j	Creates a SAS graph from the Neo4j graph database.
from_networkx	Converts a NetworkX graph to its equivalent SAS graph.
from_pandas_dataframes	Constructs a graph from the Pandas nodes or edges DataFrames or both.
from_pandas_edges	Constructs a graph from a Pandas DataFrame edge list.
from_pickle	Loads the pickled representation of a graph object from a file.
from_rdf	Converts Resource Description Framework (RDF) data to a graph object by using the RDFLib package.
from_sas_edges	Constructs a graph from a SAS data set that is stored as a file.

Function Name	Description
from_torch_geometric	Converts a *torch_geometric.data.Data* instance to a SAS graph object.
node_attrs_to_torch_tensor	Returns node attributes of the graph in the form of a torch tensor.
to_adjacency_matrix	Returns the graph adjacency matrix as a sparse or dense matrix.
to_dgl	Converts a SAS graph to a *DGL* graph.
to_directed	Returns a directed copy of the input graph.
to_dot	Produces the DOT representation of a graph.
to_gds	Creates a Neo4j in-memory graph object equivalent to the input graph.
to_graphtool	Converts a SAS graph to its equivalent graph-tool graph.
to_igraph	Converts a SAS graph to its equivalent igraph graph.
to_immutable	Creates an immutable graph from the input graph.
to_index_graph	Maps the nodes of the graph to integers.
to_molecule	Converts a SAS graph to its *rdkit* or SMILES molecule.
to_mutable	Creates a mutable graph from the input graph.
to_neo4j	Creates a graph in the Neo4j graph database equivalent to the input SAS graph.
to_networkx	Converts a SAS graph to its equivalent NetworkX graph.
to_pickle	Writes the graph in Python pickle format.
to_torch_geometric	Converts a SAS graph to *torch_geometric.data.Data*.
to_undirected	Returns a directed copy of the input graph.

edge_attrs_to_torch_tensor

Returns edge attributes of the graph in the form of a torch tensor.

API: edge_attrs_to_torch_tensor

sasviya.network.utils.conversions.edge_attrs_to_torch_tensor(g, edge_attrs)

Parameters

g : SAS graph object
Specifies the input graph to assess.

edge_attrs : str or iterable of str or 'ALL'
Specifies the edge attributes to include. The value can be a single string, a list of strings, or 'ALL' to include all edge attributes. If any of the specified edge attributes are not numeric, a ValueError is raised.

Returns

torch.Tensor
Returns a torch tensor of the shape (number of edges, number of edge attributes).

Examples

```
import dgl
import torch
g = dgl.data.QM9EdgeDataset()[0][0]
G = sasnet.from_dgl(g,create_using=sasnet.ImmutableGraph)
print(sasnet.edge_attr_to_torch_tensor(G,edge_attrs='ALL'))
```

from_dgl

Converts a *DGL graph* instance to its corresponding SAS graph.

API: from_dgl

sasviya.network.utils.conversions.from_dgl(g, node_attrs='ALL', edge_attrs='ALL', create_using=None, edge_key=None, selfloops=None)

Parameters

g : DGL graph object
Specifies the input DGL graph object to convert.

node_attrs : str or iterable of str, optional
Specifies a node attribute or list of node attributes to copy to the output graph. The default is 'ALL'.

edge_attrs : str or iterable of str, optional
Specifies an edge attribute or list of edge attributes to copy to the output graph. The default is 'ALL'.

create_using : graph type, optional
Specifies the graph type to create. By default, an instance of the MultiDiGraph class is created.

edge_key : str, optional
Specifies the edge attribute to consider as the edge key for multigraphs.

selfloops : bool or None, optional
Specifies whether or not to retain self-loops. The default is None, which means that self-loops are removed from simple graphs and retained in multigraphs.

Returns

SAS graph object.
Returns the converted SAS graph.

Examples

```
import dgl
g = dgl.data.CoraGraphDataset()[0]
G = sasnet.from_dgl(g)
```

from_gds

Creates a SAS graph equivalent to the input Neo4j in-memory graph.

API: from_gds

```
sasviya.network.utils.conversions.from_gds(g, gds, node_attrs='ALL',
edge_attrs='ALL', edge_key=None)
```

Parameters

g : SAS graph object
Specifies the input graph to convert to a Neo4j graph.

gds : GraphDataScience object or None, optional
Specifies an instance of the GraphDataScience class. The default is None.

node_attrs : str or iterable of str or None, optional
Specifies a node attribute or list of node attributes to read from the Neo4j in-memory graph. The default is 'ALL'.

edge_attrs : str or iterable of str or None, optional
Specifies an edge attribute or list of edge attributes to read from the Neo4j in-memory graph. The default is 'ALL'.

edge_key : str or None, optional
Specifies the column to interpret as the edge key in multigraphs. The default is None.

Returns

SAS graph object
Returns a SAS counterpart graph.

from_graphtool

Converts a graph-tool graph object to its equivalent SAS graph.

API: from_graphtool

```
sasviya.network.utils.conversions.from_graphtool(g, create_using=None, *,
data=True)
```

Parameters

g : graph-tool graph object
Specifies the input graph to assess.

create_using : graph type, optional
Specifies the graph type to create. By default, a multigraph is created; that is, an instance of a MultiDiGraph class if *g* is directed, and otherwise an instance of a MultiGraph class.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns the converted graph.

from_igraph

Converts an igraph graph to its equivalent SAS graph.

API: from_igraph

```
sasviya.network.utils.conversions.from_igraph(g_ig)
```

Parameters

g_ig : igraph graph object
Specifies the input graph to convert.

Returns

SAS graph object
Returns the converted graph.

from_molecule

Converts a SMILES string or *rdkit* molecule to a SAS graph.

API: from_molecule

```
sasviya.network.utils.conversions.from_molecule(mol, *, immutable=False,
add_hydrogens=False, suppress_warnings=False)
```

Parameters

mol : str or rdkit molecule object
Specifies the input molecule to convert.

immutable : bool, optional
Specifies whether or not the output graph is immutable. The default is False.

add_hydrogens : bool, optional
Specifies whether or not to include the molecule's hydrogen atoms. The default is False.

suppress_warnings : bool, optional
Specifies whether or not to suppress the warnings that the rdkit package generates. The default is False.

Returns

SAS graph object
Returns the SAS counterpart graph.

Examples

```
smiles_str = 'CN1C=NC2=C1C(=O)N(C(=O)N2C)C'
G1 = sasnet.from_molecule(smiles_str)
rdkit_mol = Chem.MolFromSmiles(smiles_str)
G2 = sasnet.from_molecule(rdkit_mol).
```

from_neo4j

Creates a SAS graph from the Neo4j graph database.

API: from_neo4j

sasviya.network.utils.conversions.from_neo4j(gds, node_col=None, node_attrs='ALL', edge_attrs='ALL', edge_key=None, multigraph=None)

Parameters

gds : GraphDataScience object or None, optional
Specifies an instance of the GraphDataScience class. The default is None.

node_col : str or None, optional
Specifies the node attribute to use as the node column. The default is None, which means that the ID (node) is used as the node identifier.

node_attrs : str or iterable of str, optional
Specifies a node attribute or list of node attributes to read from the Neo4j in-memory graph. The default is 'ALL'.

edge_attrs : str, optional
Specifies an edge attribute or list of edge attributes to read from the Neo4j in-memory graph. The default is 'ALL'.

edge_key : str or None, optional
Specifies the column to interpret as the edge key in multigraphs. The default is None.

multigraph : bool or None, optional
Specifies whether or not to keep multiedges. The default is None, which means that multiedges are removed. This parameter has no effect when the *edge_key* parameter is specified.

Returns

SAS graph object
Returns a SAS counterpart graph.

from_networkx

Converts a NetworkX graph to its equivalent SAS graph.

API: from_networkx

```
sasviya.network.utils.conversions.from_networkx(g_nx,
node_attr_handler=None, edge_attr_handler=None, *, data=True)
```

Parameters

g_nx : NetworkX graph object
Specifies the input graph to convert.

node_attr_handler : function or None, optional
Specifies a function to apply to node attributes before setting the attributes. The function should receive data as input and return the modified data.

edge_attr_handler : function or None, optional
Specifies a function to apply to edge attributes before setting the attributes. The function should receive data as input and return the modified data.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, which means that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns the converted graph.

Examples

```
import networkx as nx
g_nx = nx.Graph([[0,1,{}],[1,2,{'attr':['x','y']}]])
my_func = lambda data: {'attr': ''.join(data['attr']) if 'attr' in data else ''}
G = sasnet.from_networkx(g_nx,edge_attr_handler=my_func)
print(G.edges(data=True)) #[(0, 1, {'attr': ''}), (1, 2, {'attr': 'x,y'})].
```

from_pandas_dataframes

Constructs a graph from the Pandas nodes or edges DataFrames or both.

API: from_pandas_dataframes

sasviya.network.utils.conversions.from_pandas_dataframes(*, edges_df=None, nodes_df=None, from_col='from', to_col='to', node='node', edge_attr=None, edge_key=None, node_attr=None, create_using=None, _skip_validation=False)

Parameters

edges_df : DataFrame
Specifies the input Pandas DataFrame that contains the edge list.

nodes_df : DataFrame
Specifies the input Pandas DataFrame that contains the node list.

from_col : str, optional
Specifies the name of the column that contains the source of each edge. The default is 'from'.

to_col : str, optional
Specifies the name of the column that contains the destination of each edge. The default is 'to'.

node : str, optional
Specifies the name of the column that contains the node identifier. The default is 'node'.

edge_attr : bool or list, optional
Specifies whether or not to interpret all columns, other than the columns that you specify in the *from_col* and *to_col* parameters, as edge attributes. If you specify a list, the columns in the list are interpreted as edge attributes. The default is None, which means that no edge attributes are extracted from the Pandas DataFrame.

edge_key : str or None, optional
Specifies the column to interpret as the edge key in multigraphs. The default is None.

node_attr : bool or list, optional
Specifies whether or not to interpret all columns other than node as node attributes. If you specify a list, the columns in the list are interpreted as node attributes. The default is None, which means that no node attributes are extracted from the Pandas DataFrame.

create_using : graph type, optional
Specifies the graph type to create. By default, an instance of an ImmutableGraph class is created.

Returns

SAS graph object
Returns the computed SAS graph object.

from_pandas_edges

Constructs a graph from a Pandas DataFrame edge list.

API: from_pandas_edges

```
sasviya.network.utils.conversions.from_pandas_edges(edges_df,
from_col='from', to_col='to', edge_attr=None, create_using=None,
edge_key=None)
```

Parameters

edges_df : DataFrame
Specifies the input Pandas DataFrame.

from_col : str, optional
Specifies the name of the column that contains the source of each edge. The default is 'from'.

to_col : str, optional
Specifies the name of the column that contains the destination of each edge. The default is 'to'.

edge_attr : bool or list, optional
Specifies whether or not to interpret all columns, other than the columns that you specify in the *from_col* and *to_col* parameters, as edge attributes. If you specify a list, the columns in the list are interpreted as edge attributes. The default is None, which means that no edge attributes are extracted from the Pandas DataFrame.

create_using : graph type, optional
Specifies the graph type to create. By default, an instance of an ImmutableGraph class is created.

edge_key : str or None, optional
Specifies the column to interpret as the edge key in multigraphs. The default is None.

Returns

SAS graph object
Returns the computed graph.

from_pickle

Loads the pickled representation of a graph object from a file.

API: from_pickle

```
sasviya.network.utils.conversions.from_pickle(filename, **kwargs)
```

Parameters

filename : str
Specifies the file name to load.

Returns

SAS graph object
Returns the graph that is loaded from the pickled representation.

from_rdf

Converts Resource Description Framework (RDF) data to a graph object by using the RDFLib package.

API: from_rdf

```
sasviya.network.utils.conversions.from_rdf(file, input_format=None,
create_using=None)
```

Parameters

file : str
Specifies the RDF file to convert.

input_format : str or None, optional
Specifies the parser to use. For example, the value can be 'xml', 'n3', or 'trix'. For more information, see *https://rdflib.readthedocs.io/en/stable/intro_to_parsing.html*. The default is None, which means that the format is based on the file extension.

create_using : graph type or None, optional
Specifies the graph type to create. The default is None, which means that an instance of a MultiDiGraph class is created.

Returns

SAS graph object
Returns the converted SAS graph.

from_sas_edges

Constructs a graph from a SAS data set that is stored as a file.

API: from_sas_edges

```
sasviya.network.utils.conversions.from_sas_edges(filename, *, from_col='from',
to_col='to', edge_attr=None, create_using=None, edge_key=None,
file_type='CSV', header=0)
```

Parameters

filename : str or path object or file-like object
Specifies the SAS data set to read.

from_col : str, optional
Specifies the name of the column that contains the source of each edge. The default is 'from'.

to_col : str, optional
Specifies the name of the column that contains the destination of each edge. The default is 'to'.

edge_attr : bool or list or None, optional
Specifies whether or not to interpret all columns, other than the columns that you specify in the *from_col* and *to_col* parameters, as edge attributes. When you specify a list, the columns in the list are interpreted as edge attributes. By default, no edge attributes are extracted from the SAS data set.

create_using : graph type or None, optional
Specifies the graph type to create. By default, an instance of an ImmutableGraph class is created.

edge_key : str or None, optional
Specifies the column to interpret as the edge key in multigraphs. The default is None.

file_type : str, optional
Specifies the file format to use. The default is "CSV". The following file types are allowed. 'AUTO', 'HDAT', 'CSV', 'EXCEL', 'JMP', 'SPSS', 'DTA', 'ESP', 'LASR', 'BASESAS', 'XLS', 'FMT', 'DOCUMENT', 'IMAGE', 'AUDIO', 'VIDEO', 'ANY', 'PARQUET'.

header : int, optional
Specifies the row index containing column labels. The default value is 0. This parameter only applies when file_type is 'CSV' or 'EXCEL' or 'XLS'.

Returns

SAS graph object
Returns the converted graph.

Examples

```
filename = "test.sas7bdat"
g_out = sasnet.from_sas_edges(filename , file_type="BASESAS", create_using=sasnet.Immutab
```

from_torch_geometric

Converts a *torch_geometric.data.Data* instance to a SAS graph object.

API: from_torch_geometric

```
sasviya.network.utils.conversions.from_torch_geometric(data, node_attrs='ALL',
edge_attrs='ALL', graph_attrs='ALL', create_using=None, edge_key=None,
selfloops=None, map_func=None)
```

Parameters

data : torch_geometric.data.Data object
Specifies the input torch_geometric data object to convert.

node_attrs : str or iterable of str, optional
Specifies a node attribute or list of node attributes to copy to the output graph. The default is 'ALL'.

edge_attrs : str or iterable of str, optional
Specifies an edge attribute or list of edge attributes to copy to the output graph. The default is 'ALL'.

graph_attrs : str or iterable of str, optional
Specifies a graph-level attribute or list of attributes to copy to the output graph. The default is 'ALL'.

create_using : graph type, optional
Specifies the graph type to create. By default, an instance of a DiGraph class is created.

edge_key : str, optional
Specifies the edge attribute to consider as the edge key for multigraphs.

selfloops : bool or None, optional
Specifies whether or not to retain self-loops. The default is None, which means that self-loops are removed from simple graphs and retained in multigraphs.

map_func : function or dict of functions, optional
Specifies a map function to apply to node or edge attributes and return a dictionary of keys and values. It is also possible to pass different functions for different attributes that are keyed with the attribute name {'x':func1, 'y':func2}. If the *map_func* parameter is not specified (or is not specified for a particular

attribute), auto-unpacking rules are applied. Auto-unpacking is designed to handle standard data structures that hold numerical or string values, such as dictionary, tensor, ndarray, nested lists, or Pandas Series and DataFrames.

Returns

SAS graph object
Returns a SAS counterpart graph object.

Examples

```
import sasviya.network as sasnet
from torch_geometric.datasets import Planetoid
dataset = Planetoid('/bignetwork/user/brrees/data/pyg/', 'cora')
G = sasnet.from_torch_geometric(dataset[0], create_using=sasnet.Graph).
```

node_attrs_to_torch_tensor

Returns node attributes of the graph in the form of a torch tensor.

API: node_attrs_to_torch_tensor

sasviya.network.utils.conversions.node_attrs_to_torch_tensor(g, node_attrs)

Parameters

g : SAS graph object
Specifies the input graph to assess.

node_attrs : str or iterable of str or 'ALL'
Specifies the node attributes to include. The value can be a single string, a list of strings, or 'ALL' to include all node attributes. If any of the specified node attributes are not numeric, a ValueError is raised.

Returns

torch.Tensor
Returns a torch tensor of the size (number of nodes, number of node attributes).

Examples

```
import dgl
import torch
g = dgl.data.CoraGraphDataset()[0]
G = sasnet.from_dgl(g,create_using=sasnet.ImmutableGraph)
print(sasnet.node_attr_to_torch_tensor(G,node_attrs='ALL'))
```

to_adjacency_matrix

Returns the graph adjacency matrix as a sparse or dense matrix.

API: to_adjacency_matrix

sasviya.network.utils.conversions.to_adjacency_matrix(g, weight=None, dtype=None, *, sparse=True)

Parameters

g : SAS graph object
Specifies the input graph to assess.

weight : str or None, optional
Specifies an edge attribute that holds values to be represented in the adjacency matrix. The default is None.

dtype : NumPy dtype or None, optional
Specifies the format for initializing the adjacency matrix. The default is None.

sparse : bool, optional
Specifies whether or not to return a scipy sparse matrix. When the value is False, the result is converted to a dense format (a NumPy 2-D array). The default is True.

Returns

scipy sparse matrix or a NumPy 2-D array
Returns the adjacency matrix of the input graph.

to_dgl

Converts a SAS graph to a *DGL* graph.

API: to_dgl

```
sasviya.network.utils.conversions.to_dgl(g, node_attrs='ALL', edge_attrs='ALL',
group_node_attrs=None, group_edge_attrs=None)
```

Parameters

g : SAS graph object
Specifies the input graph to convert.

node_attrs : str or iterable of str, optional
Specifies a node attribute or list of node attributes to copy to the output graph. The default is 'ALL'.

edge_attrs : str or iterable of str, optional
Specifies an edge attribute or list of edge attributes to copy to the output graph. The default is 'ALL'.

group_node_attrs : iterable of str or 'ALL' or None, optional
Specifies a list of node attributes (or 'ALL') to concatenate and add to *ndata.feat*. The default is None.

group_edge_attrs : str or iterable of str or None, optional
Specifies a list of edge attributes (or 'ALL') to concatenate and add to *edata.feat*. The default is None.

Returns

DGL graph object
Returns a DGL graph equivalent to the input graph.

to_directed

Returns a directed copy of the input graph.

API: to_directed

```
sasviya.network.utils.conversions.to_directed(g)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

Returns

SAS graph object
Returns a directed copy of the input graph.

to_dot

Produces the DOT representation of a graph.

API: to_dot

```
sasviya.network.utils.conversions.to_dot(g, nodes_label=None,
nodes_shape='circle', nodes_size=None, nodes_style=None, nodes_size_scale=1,
nodes_color=None, nodes_color_by=None, node_attrs=None, nodes_attrs=None,
edges_label=None, edges_color=None, edges_style=None,
edges_color_by=None, edge_attrs=None, edges_attrs=None, graph_attrs=None,
output_file=None, size=10, layout=None, *, _return_object=False)
```

For more information, see https://graphviz.org/doc/info/attrs.html.

Parameters

g : SAS graph object
Specifies the input graph to assess.

nodes_label : str or None, optional
Specifies the name of an attribute to use as the node label. The default is None.

nodes_shape : str, optional
Specifies the shape of nodes in the graph. The default is 'circle'.

nodes_size : numeric or None, optional
Specifies the size of nodes in the graph. The default is None.

nodes_style : str or None, optional
Specifies the style descriptor to apply to all nodes. The default is None.

nodes_size_scale : int, optional
Specifies a scale factor to apply to each per-node size attribute. The default is 1.

nodes_color : str or None, optional
Specifies the name of an attribute to use as the node color. The default is None.

nodes_color_by : str or None, optional
Specifies the name of an attribute to map to node colors. The default is None.

node_attrs : dict or None, optional
Specifies a dictionary that maps Graphviz-supported node attributes to their respective graph node attribute names. The default is None.

nodes_attrs : dict or None, optional
Specifies a dictionary that maps Graphviz-supported node attributes to a constant value. The dictionary provides a default to all nodes unless you specify a node-specific value for this parameter. The default is None.

edges_label : str or None, optional
Specifies the name of an attribute to use as the edge label. The default is None.

edges_color : str or None, optional
Specifies the name of an attribute to use as the edge color. The default is None.

edges_style : str or None, optional
Specifies the style descriptor to apply to all edges. The default is None.

edges_color_by : str or None, optional
Specifies the name of an attribute to map to edge colors. The default is None.

edge_attrs : dict or None, optional
Specifies a dictionary that maps Graphviz-supported edge attributes to their respective graph edge attribute names. The default is None.

edges_attrs : dict or None, optional
Specifies a dictionary that maps Graphviz-supported edge attributes to a constant value. The dictionary provides a default to all edges unless you specify an edge-specific value for this parameter. The default is None.

graph_attrs : dict or None, optional
Specifies a dictionary that maps Graphviz-supported graph attributes to a constant value. The default is None.

output_file : str or None, optional
Specifies the path to an output file to write the DOT representation. The default is None.

size : int, optional
Specifies the desired size of the rendered graph. The default is 10.

layout : str, optional
Specifies the desired layout of the rendered graph. By default, no layout is included in the DOT representation.

Returns

str
Returns the graph that is represented in DOT format.

to_gds

Creates a Neo4j in-memory graph object equivalent to the input graph.

API: to_gds

```
sasviya.network.utils.conversions.to_gds(g, graph_name, gds, concurrency=4,
node_label=None, edge_label=None, node_attrs='ALL', edge_attrs='ALL', *,
replace=False)
```

For more information, see https://neo4j.com/docs/graph-data-science-client/current/.

Parameters

g : SAS graph object
Specifies the input graph to convert to a Neo4j graph.

graph_name : str
Specifies the name of the graph inside the Neo4j graph database. If a graph with the same name already exists, it is overwritten.

gds : GraphDataScience object or None, optional
Specifies an instance of the GraphDataScience class. The default is None.

concurrency : int, optional
Specifies the parallelism with which the algorithm is executed. The default is 4.

node_label : str or None, optional
Specifies the node attribute to use for labeling nodes. The default is None.

edge_label : str or None, optional
Specifies the edge attribute to use for labeling the relationship types. The default is None.

node_attrs : str or iterable of str, optional
Specifies a node attribute or list of node attributes to load into the Neo4j graph. The default is 'ALL'.

edge_attrs : str, optional
Specifies an edge attribute or list of edge attributes to load into the Neo4j graph. The default is 'ALL'.

replace : bool, optional
Specifies whether or not a table with the same name that exists in memory is removed first. The default is False.

Returns

Neo4j in-memory graph object
Returns the graph that is loaded into Neo4j.

to_graphtool

Converts a SAS graph to its equivalent graph-tool graph.

API: to_graphtool

sasviya.network.utils.conversions.to_graphtool(g, node_attr_dict=None, edge_attr_dict=None)

Parameters

g : SAS graph object
Specifies the input graph to assess.

node_attr_dict : dict or None, optional
Specifies a dictionary in which each key is the name of a node attribute in *g* and each value is a string that indicates the graph-tool type (such as 'string', 'int', etc.) of that attribute. The default is None.

edge_attr_dict : dict or None, optional
Specifies a dictionary in which each key is the name of an edge attribute in *g* and each value is a string that indicates the graph-tool type (such as 'string', 'int', etc.) of that attribute. The default is None.

Returns

graph-tool graph object
Returns the converted graph.

dict
Returns a dictionary that maps each node attribute to a graph-tool vertex PropertyMap.

dict
Returns a dictionary that maps each edge attribute to a graph-tool edge PropertyMap.

to_igraph

Converts a SAS graph to its equivalent igraph graph.

API: to_igraph

```
sasviya.network.utils.conversions.to_igraph(g)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

Returns

igraph graph object
Returns the converted graph.

to_immutable

Creates an immutable graph from the input graph.

API: to_immutable

```
sasviya.network.utils.conversions.to_immutable(g)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

Returns

SAS graph object
Returns an immutable SAS graph object.

to_index_graph

Maps the nodes of the graph to integers.

API: to_index_graph

```
sasviya.network.utils.conversions.to_index_graph(g, offset=0, map_attr=None,
immutable=None)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

offset : int, optional
Specifies the index offset for node labels. The default is 0. When offset = r, nodes are labeled r, r+1, and so on.

map_attr : str or None, optional
Specifies the node attribute name in which to store the node of the original graph. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

Returns

SAS graph object
Returns a SAS graph object with a standardized label.

to_molecule

Converts a SAS graph to its *rdkit* or SMILES molecule.

API: to_molecule

sasviya.network.utils.conversions.to_molecule(g, *, return_smiles=False)

Parameters

g : SAS graph object
Specifies the input graph to convert.

return_smiles : bool, optional
Specifies whether or not to return the SMILES string representation of a molecule. When the value is True, an rdkit mol object is returned. The default is False.

Returns

SMILES string or rdkit mol object
Returns a molecule representation of the input graph.

Examples

```
smiles_str = 'CN1C=NC2=C1C(=O)N(C(=O)N2C)C'
G1 = sasnet.from_molecule(smiles_str)
rdkit_mol = sasnet.to_molecule(G1).
```

to_mutable

Creates a mutable graph from the input graph.

API: to_mutable

```
sasviya.network.utils.conversions.to_mutable(g)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

Returns

SAS graph object
Returns a mutable SAS graph object.

to_neo4j

Creates a graph in the Neo4j graph database equivalent to the input SAS graph.

API: to_neo4j

```
sasviya.network.utils.conversions.to_neo4j(g, gds, node_label=None,
edge_label=None, node_attrs='ALL', edge_attrs='ALL')
```

Parameters

g : SAS graph object
Specifies the input graph to convert to a Neo4j graph.

gds : GraphDataScience object or None, optional
Specifies an instance of the GraphDataScience class. The default is None.

node_label : str or None, optional
Specifies the node attribute to use for labeling nodes. The default is None.

edge_label : str or None, optional
Specifies the edge attribute to use for labeling the relationship types. The default is None.

node_attrs : str or iterable of str, optional
Specifies a node attribute or list of node attributes to load into the Neo4j graph. The default is 'ALL'.

edge_attrs : str or iterable of str, optional
Specifies an edge attribute or list of edge attributes to load into the Neo4j graph. The default is 'ALL'.

to_networkx

Converts a SAS graph to its equivalent NetworkX graph.

API: to_networkx

```
sasviya.network.utils.conversions.to_networkx(g, **kwargs)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

Returns

NetworkX graph object
Returns the converted graph.

to_pickle

Writes the graph in Python pickle format.

API: to_pickle

```
sasviya.network.utils.conversions.to_pickle(g, filename, protocol=5, **kwargs)
```

Parameters

g : SAS graph object
Specifies the input graph to convert to Python pickle format.

filename : str
Specifies the file name to write. File names that end in *.gz*, *.bz2*, *.zip*, *.xz*, *.zst*, *.tar*, *.tar.gz*, *.tar.xz*, or *.tar.bz2* are compressed.

protocol : int, Optional
Specifies the pickling protocol to use. The default is *pickle.HIGHEST_PROTOCOL*.

to_torch_geometric

Converts a SAS graph to *torch_geometric.data.Data*.

API: to_torch_geometric

```
sasviya.network.utils.conversions.to_torch_geometric(g, node_attrs='ALL',
edge_attrs='ALL', graph_attrs='ALL', group_node_attrs=None,
group_edge_attrs=None, group_graph_attrs=None, handle_attr_not_exist='error',
replace_na=None)
```

Parameters

g : SAS graph object
Specifies the input graph to convert.

node_attrs : str or iterable of str, optional
Specifies a node attribute or list of node attributes to copy to the output graph. The default is 'ALL'.

edge_attrs : str or iterable of str, optional
Specifies an edge attribute or list of edge attributes to copy to the output graph. The default is 'ALL'.

graph_attrs : str or iterable of str, optional
Specifies a graph-level attribute or list of attributes to copy to the output graph. The default is 'ALL'.

group_node_attrs : str or iterable of str or 'ALL' or None, optional
Specifies a list of node attributes (or 'ALL') to concatenate and add to *data.x*. The default is None.

group_edge_attrs : str or iterable of str or 'ALL' or None, optional
Specifies a list of edge attributes (or 'ALL') to concatenate and add to *data.edge_attr*. The default is None.

group_graph_attrs : str or iterable of str or 'ALL' or None, optional
Specifies a list of graph attributes (or 'ALL') to concatenate and add to *data.graph_attr*. The default is None.

handle_attr_not_exist : str
Specifies how to handle the situation in which the requested attribute does not exist. The default is 'error'. When you specify 'error', a KeyError is raised. When

you specify 'ignore', the error is silently ignored. When you specify 'zeros', a tensor of appropriate size filled with zeros is created.

replace_na : float
Specifies the number to replace with nan values.

Returns

torch_geometric.data.Data object
Returns the PyTorch Geometric representation of the graph.

to_undirected

Returns a directed copy of the input graph.

API: to_undirected

```
sasviya.network.utils.conversions.to_undirected(g)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

Returns

SAS graph object
Returns a directed copy of the input graph.

27

Preprocessing Utilities

Functions

Function Name	Description
fit_transform_edge_attribute	Fits and applies a preprocessor to transform edge attributes.
fit_transform_node_attribute	Fits and applies a preprocessor to transform node attributes.
transform_edge_attribute	Applies a preprocessor to transform edge attributes.
transform_node_attribute	Applies a preprocessor to transform node attributes.

fit_transform_edge_attribute

Fits and applies a preprocessor to transform edge attributes.

API: fit_transform_edge_attribute

sasviya.network.utils.preprocessing.fit_transform_edge_attribute(g, edge_attr, preprocessor, new_attrs=None, *, inplace=False, drop=False, sep=False, **fit_params)

Parameters

g : SAS graph object
Specifies the input graph to assess.

edge_attr : str
Specifies the edge attributes to transform.

preprocessor : transformer object
Specifies an instance of a class that implements fitting and transform, such as *sklearn.preprocessing.OneHotEncoder*.

new_attrs : str or list of str or callable, optional
Specifies the names of the transformed attributes. By default, attribute names are inferred. When *new_attrs* is a string (str), it names a single attribute (for preprocessors that create one output attribute). When *new_attrs* is a list, the element names one or more attributes (for preprocessors that create multiple output attributes). When *new_attrs* is callable, it specifies a function that creates a name for each attribute. The function must accept three positional arguments and return a string: *new_attrs(attr, category, i)* provides a name for the ith attribute, which corresponds to the categorical value *category*, and the input attribute name *attr*.

inplace : bool, optional
Specifies whether or not to add the transformed attributes from the input graph directly. The default is False. When the value is True, a copy of the graph is returned with the transformed attributes added.

drop : bool, optional
Specifies whether or not to remove the input attribute *attr* from the output. The default is False.

sep : str or None
Specifies the separator to be used to split the string. The default is None. This parameter is required by preprocessors such as *sklearn.preprocessors.MultiLabelBinarizer* that operate on a list, and therefore each string attribute value must be split into a list.

Returns

SAS graph object or None
If the *inplace* parameter value is False, a copy of the input graph with transformed edge attributes is returned; otherwise a value of None is returned.

fit_transform_node_attribute

Fits and applies a preprocessor to transform node attributes.

API: fit_transform_node_attribute

```
sasviya.network.utils.preprocessing.fit_transform_node_attribute(g, node_attr,
preprocessor, new_attrs=None, *, inplace=False, drop=False, sep=False,
**fit_params)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

node_attr : str
Specifies the node attributes to transform.

preprocessor : transformer object
Specifies an instance of a class that implements fitting and transform, such as *sklearn.preprocessing.OneHotEncoder*.

new_attrs : str or list of str or callable, optional
Specifies the names of the transformed attributes. By default, attribute names are inferred. When *new_attrs* is a string (str), it names a single attribute (for preprocessors that create one output attribute). When *new_attrs* is a list, the element names one or more attributes (for preprocessors that create multiple output attributes). When *new_attrs* is callable, it specifies a function that creates a name for each attribute. The function must accept three positional arguments and return a string: *new_attrs(attr, category, i)* provides a name for the ith attribute, which corresponds to the categorical value *category*, and the input attribute name *attr*.

inplace : bool, optional
Specifies whether or not to add the transformed attributes from the input graph directly. The default is False. When the value is True, a copy of the graph is returned with the transformed attributes added.

drop : bool, optional
Specifies whether or not to remove the input attribute *attr* from the output. The default is False.

sep : str or None, optional
Specifies the separator to be used to split the string. The default is None. This parameter is required by preprocessors such as *sklearn.preprocessors.MultiLabelBinarizer* that operate on a list, and therefore each string attribute value must be split into a list.

Returns

SAS graph object or None
If the *inplace* parameter value is False, a copy of the input graph with transformed node attributes is returned; otherwise a value of None is returned.

transform_edge_attribute

Applies a preprocessor to transform edge attributes.

API: transform_edge_attribute

```
sasviya.network.utils.preprocessing.transform_edge_attribute(g, edge_attr,
preprocessor, new_attrs=None, *, inplace=False, drop=False, sep=False,
**fit_params)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

edge_attr : str
Specifies the edge attributes to transform.

preprocessor : transformer object
Specifies an instance of a class that implements fitting and transform, such as *sklearn.preprocessing.OneHotEncoder*. If the preprocessor requires fitting, it should already be fitted. Otherwise, you should instead use *fit_transform_edge_attribute()*.

new_attrs : str or list of str or callable, optional
Specifies the names of the transformed attributes. By default, attribute names are inferred. When *new_attrs* is a string (str), it names a single attribute (for preprocessors that create one output attribute). When *new_attrs* is a list, the element names one or more attributes (for preprocessors that create multiple output attributes). When *new_attrs* is callable, it specifies a function that creates a name for each attribute. The function must accept three positional arguments and return a string: *new_attrs(attr, category, i)* provides a name for the ith attribute, which corresponds to the categorical value *category*, and the input attribute name *attr*.

inplace : bool, optional
Specifies whether or not to add the transformed attribute from the input graph directly. The default is False. When the value is true, a copy of the graph is returned with the transformed attributes added.

drop : bool, optional
Specifies whether or not to remove the input attribute *attr* from the output. The default is False.

sep : str or None
Specifies the separator to be used to split the string. The default is None. This parameter is required by preprocessors such as *sklearn.preprocessors.MultiLabelBinarizer* that operate on a list, and therefore each string attribute value must be split into a list.

Returns

SAS graph object or None
If the *inplace* parameter value is False, a copy of the input graph with transformed edge attributes is returned; otherwise a value of None is returned.

transform_node_attribute

Applies a preprocessor to transform node attributes.

API: transform_node_attribute

sasviya.network.utils.preprocessing.transform_node_attribute(g, node_attr, preprocessor, new_attrs=None, *, inplace=False, drop=False, sep=False, **fit_params)

Parameters

g : SAS graph object
Specifies the input graph to assess.

node_attr : str
Specifies the node attributes to transform.

preprocessor : transformer object
Specifies an instance of a class that implements fitting and transform, such as *sklearn.preprocessing.OneHotEncoder*. If the preprocessor requires fitting, it should already be fitted. Otherwise, you should instead use *fit_transform_node_attribute()*.

new_attrs : str or list of str or callable, optional
Specifies the names of the transformed attributes. By default, attribute names are inferred. When *new_attrs* is a string (str), it names a single attribute (for preprocessors that create one output attribute). When *new_attrs* is a list, the element names one or more attributes (for preprocessors that create multiple output attributes). When *new_attrs* is callable, it specifies a function that creates a name for each attribute. The function must accept three positional arguments and return a string: *new_attrs(attr, category, i)* provides a name for the ith attribute, which corresponds to the categorical value *category*, and the input attribute name *attr*.

inplace : bool, optional
Specifies whether or not to add the transformed attribute from the input graph directly. The default is False. When the value is True, a copy of the graph is returned with the transformed attributes added.

drop : bool, optional
Specifies whether or not to remove the input attribute *attr* from the output. The default is False.

sep : str or None
Specifies the separator to be used to split the string. The default is None. This parameter is required by preprocessors such as *sklearn.preprocessors.MultiLabelBinarizer* that operate on a list, and therefore each string attribute value must be split into a list.

Returns

SAS graph object or None
If the *inplace* parameter value is False, a copy of the input graph with transformed node attributes is returned; otherwise a value of None is returned.

Examples

```
import sasviya.network as sasnet
from sklearn.preprocessing import OneHotEncoder
G = sasnet.from_molecule('NC1=NC(=O)N(C=C1)[C@@H]2O[C@H](CO)[C@@H](O)C2(F)F')
ohe = OneHotEncoder(categories=[[6,7,8,9,10]], handle_unknown='ignore')
sasnet.fit_transform_node_attribute(G, 'atomic_num', ohe,inplace=True)
print(G.nodes(data=True))
```

28

Random Attribute Generator Utilities

Functions

Function Name	Description
rand_edge_attr_float	Adds a random float as an edge attribute.
rand_edge_attr_from	Adds a randomly chosen value from a specified list as an edge attribute.
rand_edge_attr_int	Adds a random integer as an edge attribute.
rand_edge_attr_str	Adds a random integer string as an edge attribute.
rand_node_attr_float	Adds a random float as a node attribute.
rand_node_attr_from	Adds a randomly chosen value from a specified list as a node attribute.
rand_node_attr_int	Adds a random integer as a node attribute.
rand_node_attr_str	Adds a random integer string as a node attribute.

rand_edge_attr_float

Adds a random float as an edge attribute.

API: rand_edge_attr_float

```
sasviya.network.utils.random_attr_generator.rand_edge_attr_float(g, min_val=0,
max_val=1, name='rand_flt', edges_subset=None, seed=None)
```

Parameters

g : SAS graph object
Specifies the input graph to add the attribute to.

min_val : int, optional
Specifies the minimum value of the float range to include. The default is 0.

max_val : int, optional
Specifies the maximum value of the float range to include. The default is 1.

name : str, optional
Specifies the name of the attribute to add. The default is “rand_flt”.

edges_subset : Container or None, optional
Specifies the subset of edges to restrict the attribute to. The default is None.

seed : int or None, optional
Specifies the random seed for reproducibility. The default is None.

rand_edge_attr_from

Adds a randomly chosen value from a specified list as an edge attribute.

API: rand_edge_attr_from

```
sasviya.network.utils.random_attr_generator.rand_edge_attr_from(g, choices,
name='rand_item', edges_subset=None, seed=None)
```

Parameters

g : SAS graph object
Specifies the input graph to add the attribute to.

choices : list
Specifies the list of attribute values to choose from.

name : str, optional
Specifies the name of the attribute to add. The default is “rand_item”.

edges_subset : Container or None, optional
Specifies the subset of edges to restrict the attribute to. The default is None.

seed : int or None, optional
Specifies the random seed for reproducibility. The default is None.

rand_edge_attr_int

Adds a random integer as an edge attribute.

API: rand_edge_attr_int

```
sasviya.network.utils.random_attr_generator.rand_edge_attr_int(g, min_val=0,
max_val=100, name='rand_int', edges_subset=None, seed=None)
```

Parameters

g : SAS graph object
Specifies the input graph to add the attribute to.

min_val : int, optional
Specifies the minimum value of the integer range to include. The default is 0.

max_val : int, optional
Specifies the maximum value of the integer range to include. The default is 100.

name : str, optional
Specifies the name of the attribute to add. The default is “rand_int”.

edges_subset : Container or None, optional
Specifies the subset of edges to restrict the attribute to. The default is None.

seed : int or None, optional
Specifies the random seed for reproducibility. The default is None.

rand_edge_attr_str

Adds a random integer string as an edge attribute.

API: rand_edge_attr_str

```
sasviya.network.utils.random_attr_generator.rand_edge_attr_str(g, min_val=0,
max_val=10000000000, name='rand_str', edges_subset=None, seed=None)
```

Parameters

g : SAS graph object
Specifies the input graph to add the attribute to.

min_val : int, optional
Specifies the minimum value of the integer range to include. The default is 0.

max_val : int, optional
Specifies the maximum value of the integer range to include. The default is 10,000,000,000.

name : str, optional
Specifies the name of the attribute to add. The default is “rand_str”.

edges_subset : Container or None, optional
Specifies the subset of edges to restrict the attribute to. The default is None.

seed : int or None, optional
Specifies the random seed for reproducibility. The default is None.

rand_node_attr_float

Adds a random float as a node attribute.

API: rand_node_attr_float

```
sasviya.network.utils.random_attr_generator.rand_node_attr_float(g, min_val=0,
max_val=1, name='rand_flt', nodes_subset=None, seed=None)
```

Parameters

g : SAS graph object
Specifies the input graph to add the attribute to.

min_val : int, optional
Specifies the minimum value of the float range to include. The default is 0.

max_val : int, optional
Specifies the maximum value of the float range to include. The default is 1.

name : str, optional
Specifies the name of the attribute to add. The default is “rand_flt”.

nodes_subset : Container or None, optional
Specifies the subset of nodes to restrict the attribute to. The default is None.

seed : int or None, optional
Specifies the random seed for reproducibility. The default is None.

rand_node_attr_from

Adds a randomly chosen value from a specified list as a node attribute.

API: rand_node_attr_from

```
sasviya.network.utils.random_attr_generator.rand_node_attr_from(g, choices,
name='rand_item', nodes_subset=None, seed=None)
```

Parameters

g : SAS graph object
Specifies the input graph to add the attribute to.

choices : list
Specifies the list of attribute values to choose from.

name : str, optional
Specifies the name of the attribute to add. The default is “rand_item”.

nodes_subset : Container or None, optional
Specifies the subset of nodes to restrict the attribute. The default is None.

seed : int or None, optional
Specifies the random seed for reproducibility. The default is None.

rand_node_attr_int

Adds a random integer as a node attribute.

API: rand_node_attr_int

```
sasviya.network.utils.random_attr_generator.rand_node_attr_int(g, min_val=0,
max_val=100, name='rand_int', nodes_subset=None, seed=None)
```

Parameters

g : SAS graph object
Specifies the input graph to add the attribute to.

min_val : int, optional
Specifies the minimum value of the integer range to include. The default is 0.

max_val : int, optional
Specifies the maximum value of the integer range to include. The default is 100.

name : str, optional
Specifies the name of the attribute to add. The default is “rand_int”.

nodes_subset : Container or None, optional
Specifies the subset of nodes to restrict the attribute to. The default is None.

seed : int or None, optional
Specifies the random seed for reproducibility. The default is None.

rand_node_attr_str

Adds a random integer string as a node attribute.

API: rand_node_attr_str

```
sasviya.network.utils.random_attr_generator.rand_node_attr_str(g, min_val=0,
max_val=10000000000, name='rand_str', nodes_subset=None, seed=None)
```

Parameters

g : SAS graph object
Specifies the input graph to add the attribute to.

min_val : int, optional
Specifies the minimum value of the integer range to include. The default is 0.

max_val : int, optional
Specifies the maximum value of the integer range to include. The default is 10,000,000,000.

name : str, optional
Specifies the name of the attribute to add. The default is “rand_str”.

nodes_subset : Container or None, optional
Specifies the subset of nodes to restrict the attribute to. The default is None.

seed : int or None, optional
Specifies the random seed for reproducibility. The default is None.

29

Visualization Utilities

Classes

Class Name	Description
PlotGraph	Base class for all plot elements.

PlotGraph

Base class for all plot elements.

API: PlotGraph

class sasviya.network.utils.visualization.PlotGraph(g, **kwargs)

It provides functionalities to plot graphs by using the NetworkX, pyvis, or rdkit package. The PlotGraph object is instantiated by calling the PlotGraph class. You can specify the following parameters when the PlotGraph object is initialized or later by using the setter methods.

Parameters

g : SAS graph object
Specifies the input graph to plot.

Methods

Method Name	Description
get_arrow_size	Gets the arrow size (for directed graphs).
get_edges_cmap	Gets the edge colormap.
get_edges_color	Gets the colors of the edges.
get_edges_color_by_category	Gets the edge attribute to color edges, based on the attribute's distinct categories.
get_edges_color_by_value	Get the edge attribute to color edges, based on the attribute's numeric value.
get_edges_color_vrange	Gets the minimum and maximum values for edge colormap scaling.
get_edges_label	Gets the edge labels.
get_edges_label_formatter	Gets the edge label formatter.
get_edges_style	Gets the style(s) of edges.
get_edges_tooltip	Gets the edge tooltips.
get_edges_width	Gets the widths of the edges.
get_layout	Gets the layout.
get_nodes_border_color	Gets the border color of nodes.
get_nodes_cmap	Gets the node colormap.
get_nodes_color	Gets the node color.
get_nodes_color_by_category	Gets the node attribute to color nodes, based on the attribute's distinct categories.
get_nodes_color_by_value	Gets the node attribute to color nodes, based on a numeric attribute.

Method Name	Description
get_nodes_color_vrange	Gets the minimum and maximum values for node colormap scaling.
get_nodes_label	Gets the nodes label.
get_nodes_label_formatter	Gets the nodes label formatter.
get_nodes_pos	Gets the (x,y)-coordinates of nodes.
get_nodes_shape	Gets the shape(s) of nodes.
get_nodes_size	Gets the size of nodes.
get_nodes_tooltip	Gets the node tooltips.
get_nodes_x	Gets the x-coordinates of nodes.
get_nodes_y	Gets the y-coordinates of nodes.
get_seed	Gets the seed value.
save	Saves the plot to a file.
set_arrow_size	Sets the arrow size for directed graphs.
set_edges_cmap	Sets the colormaps for coloring edges.
set_edges_color	Sets the colors of the edges.
set_edges_color_by_category	Sets the edge attribute to color edges, based on the attribute's distinct categories.
set_edges_color_by_value	Sets the edge attribute to color edges, based on the attribute's numeric value.
set_edges_color_vrange	Sets the minimum and maximum values for edge colormap scaling.
set_edges_label	Sets labels to appear for each edge.
set_edges_label_formatter	Sets labels formatter to apply to the numeric edge labels.
set_edges_style	Sets the edge style.
set_edges_tooltip	Sets tooltips to appear for each edge.
set_edges_width	Sets the widths of the edges.
set_layout	Sets the plot layout.

Method Name	Description
set_nodes_border_color	Sets the border color of nodes.
set_nodes_cmap	Sets the colormaps for coloring nodes.
set_nodes_color	Sets the node colors.
set_nodes_color_by_category	Sets the node attribute to color nodes, based on the attribute's different categories.
set_nodes_color_by_value	Sets the node attribute to color nodes, based on a numeric attribute.
set_nodes_color_vrange	Sets the minimum and maximum values for node colormap scaling.
set_nodes_label	Sets labels to appear for each node.
set_nodes_label_formatter	Sets the labels formatter to apply to the numeric node labels.
set_nodes_pos	Sets the (x,y)-coordinates of each node.
set_nodes_shape	Sets the nodes shape (markers).
set_nodes_size	Sets the nodes size.
set_nodes_tooltip	Sets tooltips to appear for each node.
set_nodes_x	Sets the x-coordinate of each node.
set_nodes_y	Sets the y-coordinate of each node.
set_seed	Sets the seed value for a deterministic layout.
show	Displays the plot.

get_arrow_size

```
get_arrow_size()
```

Gets the arrow size (for directed graphs).

Returns

numeric (int or float)
Returns the size of arrows.

get_edges_cmap

```
get_edges_cmap()
```

Gets the edge colormap.

Returns

str
Returns the edge colormap.

get_edges_color

```
get_edges_color()
```

Gets the colors of the edges.

Returns

dict
Returns a dictionary in which the keys are edges and the values are colors.

get_edges_color_by_category

```
get_edges_color_by_category()
```

Gets the edge attribute to color edges, based on the attribute's distinct categories.

Returns

str
Returns an edge attribute.

get_edges_color_by_value

```
get_edges_color_by_value()
```

Get the edge attribute to color edges, based on the attribute's numeric value.

Returns

str
Returns an edge attribute.

get_edges_color_vrange

```
get_edges_color_vrange()
```

Gets the minimum and maximum values for edge colormap scaling.

Returns

tuple
Returns (min_val, max_val).

get_edges_label

```
get_edges_label()
```

Gets the edge labels.

Returns

dict
Returns a dictionary in which the keys are edges and the values are edge labels.

get_edges_label_formatter

```
get_edges_label_formatter()
```

Gets the edge label formatter.

Returns

str
Returns a string that represents the format to apply to the numeric edge label.

get_edges_style

```
get_edges_style()
```

Gets the style(s) of edges.

Returns

str or dict
Returns a string that represents edge styles or a dictionary in which the keys are edges and the values are strings that represent edge styles.

get_edges_tooltip

```
get_edges_tooltip()
```

Gets the edge tooltips.

Returns

dict
Returns a dictionary in which the keys are edges and the values are tooltips.

get_edges_width

```
get_edges_width()
```

Gets the widths of the edges.

Returns

numeric or dict
Returns a dictionary in which the keys are edges and the values are edge widths.

get_layout

```
get_layout()
```

Gets the layout.

Returns

str
Returns the layout algorithm for locating nodes.

get_nodes_border_color

```
get_nodes_border_color()
```

Gets the border color of nodes.

Returns

dict
Returns a dictionary in which the keys are nodes and the values are border colors.

get_nodes_cmap

```
get_nodes_cmap()
```

Gets the node colormap.

Returns

str
Returns the node colormap.

get_nodes_color

```
get_nodes_color()
```

Gets the node color.

Returns

dict
Returns a dictionary in which the keys are nodes and the values are node colors.

get_nodes_color_by_category

```
get_nodes_color_by_category()
```

Gets the node attribute to color nodes, based on the attribute's distinct categories.

Returns

str
Returns a node attribute.

get_nodes_color_by_value

```
get_nodes_color_by_value()
```

Gets the node attribute to color nodes, based on a numeric attribute.

Returns

str
Returns a node attribute.

get_nodes_color_vrange

```
get_nodes_color_vrange()
```

Gets the minimum and maximum values for node colormap scaling.

Returns

tuple
Returns (min_val, max_val).

get_nodes_label

```
get_nodes_label()
```

Gets the nodes label.

Returns

dict
Returns a dictionary in which the keys are nodes and the values are node labels.

get_nodes_label_formatter

```
get_nodes_label_formatter()
```

Gets the nodes label formatter.

Returns

str
Returns a string that represents the format to apply to the numeric nodes label.

get_nodes_pos

```
get_nodes_pos()
```

Gets the (x,y)-coordinates of nodes.

Returns

dict
Returns a dictionary in which the keys are nodes and the values are node positions.

get_nodes_shape

```
get_nodes_shape()
```

Gets the shape(s) of nodes.

Possible shape value: “s”,”o”,”^”,”>”,”v”,”<”,”d”,”p”,”h”,”8”.

Returns

str or dict
Returns a string that represents node shapes or a dictionary in which the keys are nodes and the values are strings that represent node shapes.

get_nodes_size

```
get_nodes_size()
```

Gets the size of nodes.

Returns

numeric or dict
Returns a numeric value that represents node sizes or a dictionary in which the keys are nodes and the values are node sizes.

get_nodes_tooltip

```
get_nodes_tooltip()
```

Gets the node tooltips.

Returns

dict
Returns a dictionary in which the keys are nodes and the values are tooltips.

get_nodes_x

```
get_nodes_x()
```

Gets the x-coordinates of nodes.

Returns

dict
Returns a dictionary in which the keys are nodes and the values are x-coordinates.

get_nodes_y

```
get_nodes_y()
```

Gets the y-coordinates of nodes.

Returns

dict
Returns a dictionary in which the keys are nodes and the values are y-coordinates.

get_seed

```
get_seed()
```

Gets the seed value.

Returns

int or None
Returns the seed value.

save

```
save(filename, mode='static', with_labels=None, figsize=(12, 8), dpi=60,
layout=None, seed=None, ax=None, scale_factor=None, title=None, *,
display=False, nodes_color_legend=False, edges_color_legend=False, **kwargs)
```

Saves the plot to a file.

Parameters

filename : str
Specifies the name of the file for saving the image in PNG or JPG format.

mode : str, optional
Specifies the plot mode ('static', 'interactive', 'mol2D', or 'mol3D'). The default is 'static'. The modes mol2D and mol3D work only for molecular graphs that can be converted using sasnet.to_molecule.

with_labels : bool or None, optional
Specifies whether or not to show labels. When the value is False, labels are not shown. The default is None, which means that labels are shown for 'static' and 'interactive' mode but not for 'mol2D' or 'mol3D'.

figsize : tuple, optional
Specifies the figure size. The default is (12, 8).

dpi : int, optional
Specifies dots per inch (or resolution). The default is 60.

layout : str or None, optional
Specifies the plot layout. The default is None. When you display the graph in 'static' or 'interactive' mode, and the *nodes_x*, *nodes_y*, or *nodes_pos* option is not specified, the 'kamada_kawai' layout is used. Setting *layout* options for displaying molecular graphs in "mol2d" or "mol3d" mode raises an error. The following layouts are supported: 'bipartite', 'circular', 'fruchterman_reingold', 'kamada_kawai', 'multipartite', 'planar', 'random', 'shell', 'spectral', 'spiral', 'spring', 'circo', 'dot', 'fdp', 'neato', 'sfdp', 'twopi'. The 'circo', 'dot', 'fdp', 'neato', 'sfdp', 'twopi' layouts require Graphviz and pydot to be installed.

seed : int, optional
Specifies the random seed.

ax : matplotlib axes object or None, optional
Specifies the axes of the current figure. This parameter is supported only when the *mode* parameter value is 'static' or 'mol2D'. The default is None.

scale_factor : float or None, optional
Specifies the factor to linearly scale the nodes position. The default is None. A value greater than 1 enlarges the graph, and a value less than 1 shrinks the graph. This parameter is supported only when the *mode* parameter value is 'static' or 'interactive'. This parameter is ignored when the *nodes_x* and *nodes_y* coordinates are set.

title : str, optional
Specifies the text to use for the title.

display : bool, optional
Specifies whether or not to display the plot. The default is False.

nodes_color_legend : bool, optional
Specifies whether or not to display a color legend when nodes are colored by either category or value; that is, when the *nodes_color_by_category* or *nodes_color_by_value* parameter is specified. The default is False. This feature is currently not supported when the *mode* parameter value is 'mol3D'.

edges_color_legend : bool, optional
Specifies whether or not to display a color legend when edges are colored by either category or value; that is, when the *edges_color_by_category* or *edges_color_by_value* parameter is specified. The default is False. This feature is currently not supported when the *mode* parameter value is 'mol3D'.

Returns

plot or None
Returns None or the plotted graph, based on the value of the *display* parameter.

set_arrow_size

```
set_arrow_size(value)
```

Sets the arrow size for directed graphs.

Parameters

value : numeric or str or list or dict
Specifies the size of arrows. When you specify a number, the size of all arrows is set to that number. When you specify a string (str), it names an edge attribute that contains the arrow sizes. When you specify a list, it contains the size of each arrow. When you specify a dictionary, the keys are edges and the values are arrow sizes.

set_edges_cmap

```
set_edges_cmap(value)
```

Sets the colormaps for coloring edges.

Parameters

value : str or None
Specifies the edge colormap. See https://matplotlib.org/stable/users/explain/colors/colormaps.html to find possible values for colormaps; for example, 'binary', 'gist_heat', 'Reds', 'RdYlGn', 'Pastel1'.

set_edges_color

```
set_edges_color(value)
```

Sets the colors of the edges.

Parameters

value : str or list or dict
Specifies the edge color. When you specify a string (str), it names an edge attribute that contains the edge color, or it represents a hex color (for example, #000000) or one of the named colors that are supported by Matplotlib (see https://matplotlib.org/3.3.1/gallery/color/named_colors.html). When you specify a list, it contains the color of each edge. When you specify a dictionary, the keys are edges and the values are colors.

set_edges_color_by_category

```
set_edges_color_by_category(value, edges_cmap=None)
```

Sets the edge attribute to color edges, based on the attribute's distinct categories.

Parameters

value : str or None
Specifies an edge attribute.

edges_cmap : str or None, optional
Specifies the colormaps for coloring edges. The default value is None. For more information about this parameter, see the *set_edges_cmap* parameter.

set_edges_color_by_value

```
set_edges_color_by_value(value, edges_cmap=None)
```

Sets the edge attribute to color edges, based on the attribute's numeric value.

Parameters

value : str or None
Specifies a numeric edge attribute.

edges_cmap : str or None, optional
Specifies the colormaps for coloring edges. The default value is None. For more information about this parameter, see the *set_edges_cmap* parameter.

set_edges_color_vrange

```
set_edges_color_vrange(value)
```

Sets the minimum and maximum values for edge colormap scaling.

Parameters

value : tuple of float or None
Specifies numeric values (min_val, max_val) that map to the minimum boundaries of the color scale.

set_edges_label

```
set_edges_label(value)
```

Sets labels to appear for each edge.

Parameters

value : str or list or dict or None
Specifies the edge labels. When you specify a string (str), it names an edge attribute that contains the edge labels. When you specify a list, it contains the label of each edge. When you specify a dictionary, the keys are edges and the values are edge labels.

set_edges_label_formatter

```
set_edges_label_formatter(value)
```

Sets labels formatter to apply to the numeric edge labels.

Parameters

value : str
Specifies the edge label format. The value can be a string formatter, such as '.5g' or '.2f'. For more information about standard formats, see https://docs.python.org/3/tutorial/inputoutput.html.

set_edges_style

```
set_edges_style(value)
```

Sets the edge style.

Parameters

value : str or list or dict
Specifies the edge style. When you specify a string (str), it represents a line style or names an edge attribute that contains the line styles. The possible styles are 'solid' or '-', 'dashed' or '–', 'dotted' or ':', and 'dashdot' or '-.'. When you specify a list, it contains the style of each edge. When you specify a dictionary, the keys are edges and the values are edge styles. Note: For interactive plots, all non-solid styles are rendered as dashed.

set_edges_tooltip

```
set_edges_tooltip(value)
```

Sets tooltips to appear for each edge.

Parameters

value : str or list or dict or None
Specifies the edge tooltip. When you specify a string (str), it names an edge attribute that contains the edge tooltips. When you specify a list, it contains the tooltip of each edge. When you specify a dictionary, the keys are edges and the values are tooltips.

set_edges_width

```
set_edges_width(value)
```

Sets the widths of the edges.

Parameters

value : numeric or str or list or dict
Specifies the edge width. When you specify a number, all edge widths are set to that number. When you specify a string (str), it names an edge attribute that contains the edge widths. When you specify a list, it contains the width of each edge. When you specify a dictionary, the keys are edges and the values are edge widths.

set_layout

```
set_layout(value)
```

Sets the plot layout.

Parameters

value : str or None
Specifies the plot layout. The following layouts are supported: 'bipartite', 'circular', 'fruchterman_reingold', 'kamada_kawai', 'multipartite', 'planar', 'random', 'shell', 'spectral', 'spiral', 'spring', 'circo', 'dot', 'fdp', 'neato', 'sfdp', and 'twopi'. The 'circo', 'dot', 'fdp', 'neato', 'sfdp', and 'twopi' layouts require Graphviz and pydot to be installed.

set_nodes_border_color

```
set_nodes_border_color(value)
```

Sets the border color of nodes.

Parameters

value : str or list or dict
Specifies the border color of nodes. When you specify a string (str), it names a node attribute that contains the border color, or it represents a hex color (for example, #000000) or one of the named colors that are supported by Matplotlib (see https://matplotlib.org/3.3.1/gallery/color/named_colors.html). When you specify a list, it contains the border color of each node. When you specify a dictionary, the keys are nodes and the values are border colors.

set_nodes_cmap

```
set_nodes_cmap(value)
```

Sets the colormaps for coloring nodes.

Parameters

value : str or None
Specifies the colormap. See https://matplotlib.org/stable/users/explain/colors/colormaps.html to find possible values for colormaps; for example, 'binary', 'gist_heat', 'Reds', 'RdYlGn', 'Pastel1'.

set_nodes_color

```
set_nodes_color(value)
```

Sets the node colors.

Parameters

value : str or list or dict
Specifies the node colors. When you specify a string (str), it names a node attribute that contains the node color, or it represents a hex color (for example, #000000) or one of the named colors that are supported by Matplotlib (see https://matplotlib.org/3.3.1/gallery/color/named_colors.html). When you specify a list, it contains the color of each node. When you specify a dictionary, the keys are nodes and the values are node colors.

set_nodes_color_by_category

```
set_nodes_color_by_category(value, nodes_cmap=None)
```

Sets the node attribute to color nodes, based on the attribute's different categories.

Parameters

value : str or None
Specifies a node attribute.

nodes_cmap : str or None, optional
Specifies the colormaps for coloring nodes. The default value is None. For more information about this parameter, see the *set_nodes_cmap* parameter.

set_nodes_color_by_value

```
set_nodes_color_by_value(value, nodes_cmap=None)
```

Sets the node attribute to color nodes, based on a numeric attribute.

Parameters

value : str or None
Specifies a numeric node attribute.

nodes_cmap : str or None, optional
Specifies the colormaps for coloring nodes. The default value is None. For more information about this parameter, see the *set_nodes_cmap* parameter.

set_nodes_color_vrange

```
set_nodes_color_vrange(value)
```

Sets the minimum and maximum values for node colormap scaling.

Parameters

value : tuple of float or None
Specifies numeric values (min_val, max_val) that map to the minimum boundaries of the color scale.

set_nodes_label

```
set_nodes_label(value)
```

Sets labels to appear for each node.

Parameters

value : str or list or dict
Specifies the node labels. When you specify a string (str), it names a node attribute that contains the node labels. When you specify a list, it contains the node label of each node. When you specify a dictionary, the keys are nodes and the values are node labels.

set_nodes_label_formatter

```
set_nodes_label_formatter(value)
```

Sets the labels formatter to apply to the numeric node labels.

Parameters

value : str
Specifies a string formatter; for example, '.5g' or '.2f'. For more information about standard formats, see https://docs.python.org/3/tutorial/inputoutput.html.

set_nodes_pos

```
set_nodes_pos(value)
```

Sets the (x,y)-coordinates of each node.

Parameters

value : str or list or dict or None
Specifies the node positions. When you specify a string (str), it names a node attribute that contains the x- and y-coordinates of each node. When you specify a list, it contains the x- and y-coordinates of each node. When you specify a dictionary, the keys are nodes and the values are x- and y-coordinates.

set_nodes_shape

```
set_nodes_shape(value)
```

Sets the nodes shape (markers).

Parameters

value : str or list or dict
Specifies the node shape. When you specify a string (str), it represents a node shape for all nodes, or it names a node attribute that contains the node shapes. The possible node shapes are 'ellipse', 'e', 'square', 's', 'circle', 'o', 'triangle', '^','triangle_up','^', 'triangle_left', '<', 'triangle_right', '>', 'triangle_down', 'v', 'pentagon', 'p', 'hexagon', and 'h'. When you specify a list, it contains the node shape of each node. When you specify a dictionary, the keys are nodes and the values are node shapes.

set_nodes_size

```
set_nodes_size(value)
```

Sets the nodes size.

Parameters

value : numeric or str or list or dictionary
Specifies the node size. When you specify a number, all node sizes are set to that number. When you specify a string (str), it names a node attribute that contains the node sizes. When you specify a list, it contains the size of each node. When you specify a dictionary, the keys are nodes and the values are node sizes.

set_nodes_tooltip

```
set_nodes_tooltip(value)
```

Sets tooltips to appear for each node.

Parameters

value : str or list or dict or None
Specifies the nodes tooltip value. When you specify a string (str), it names a node attribute that contains the node tooltips. When you specify a list, it contains the node tooltip of each node. When you specify a dictionary, the keys are nodes and the values are tooltips.

set_nodes_x

```
set_nodes_x(value)
```

Sets the x-coordinate of each node.

Parameters

value : str or list or dict or None
Specifies the x-coordinates of the nodes. When you specify a string (str), it names a node attribute that contains the x-coordinate of each node. When you specify a list, it contains the x-coordinate of each node. When you specify a dictionary, the keys are nodes and the values are x-coordinates.

set_nodes_y

```
set_nodes_y(value)
```

Sets the y-coordinate of each node.

Parameters

value : str or list or dict or None
Specifies the y-coordinate of a node. When you specify a string (str), it names a node attribute that contains the y-coordinate of each node. When you specify a list, it contains the y-coordinate of each node. When you specify a dictionary, the keys are nodes and the values are y-coordinates.

set_seed

```
set_seed(value)
```

Sets the seed value for a deterministic layout.

Parameters

value : int or None
Specifies the seed value.

show

```
show(mode='static', with_labels=None, figsize=(12, 8), dpi=60, layout=None,
seed=None, ax=None, scale_factor=None, title=None, *, display=True,
nodes_color_legend=False, edges_color_legend=False, **kwargs)
```

Displays the plot.

Parameters

mode : str, optional
Specifies the plot mode ('static', 'interactive', 'mol2D', or 'mol3D'). The default is 'static'. The modes mol2D and mol3D work only for molecular graphs that can be converted using sasnet.to_molecule.

with_labels : bool or None, optional
Specifies whether or not to show labels. When the value is False, labels are not shown. The default is None, which means that labels are shown for 'static' and 'interactive' mode but not for 'mol2D' or 'mol3D'.

figsize : tuple, optional
Specifies the figure size. The default is (12, 8). This parameter is ignored when the *ax* parameter is specified.

dpi : int, optional
Specifies dots per inch (or resolution). The default is 60. This parameter is ignored when the *ax* parameter is specified.

layout : str or None, optional
Specifies the plot layout. The default is None. When you display the graph in 'static' or 'interactive' mode, and the *nodes_x*, *nodes_y*, or *nodes_pos* option is not specified, the 'kamada_kawai' layout is used. Setting *layout* options for displaying molecular graphs in "mol2d" or "mol3d" mode raises an error. The following layouts are supported: 'bipartite', 'circo', 'circular', 'dot', 'fdp', 'fruchterman_reingold', 'kamada_kawai', 'multipartite', 'neato', 'planar', 'random', 'sfdp', 'shell', 'spectral', 'spiral', 'spring', and 'twopi'. The 'circo', 'dot', 'fdp', 'neato', 'sfdp', and 'twopi' layouts require Graphviz and pydot to be installed.

seed : int, optional
Specifies the random seed.

ax : matplotlib axes object or None, optional
Specifies the axes of the current figure. This parameter is supported when the *mode* parameter value is 'static' or 'mol2D'. The default is None.

scale_factor : float or None, optional
Specifies the factor to linearly scale the nodes position. The default is None. A value greater than 1 enlarges the graph, and a value less than 1 shrinks the graph. This parameter is supported only when the *mode* parameter value is 'static' or 'interactive'. This parameter is ignored when the *nodes_x* and *nodes_y* coordinates are set.

title : str, optional
Specifies the text to use for the title.

display : bool, optional
Specifies whether or not to display the plot and close it. The default is True.

nodes_color_legend : bool, optional
Specifies whether or not to display a color legend when nodes are colored by either category or value; that is, when the *nodes_color_by_category* or *nodes_color_by_value* parameter is specified. The default is False. This feature is not supported when the *mode* parameter value is 'mol3D'.

edges_color_legend : bool, optional
Specifies whether or not to display a color legend when edges are colored by either category or value; that is, when the *edges_color_by_category* or *edges_color_by_value* parameter is specified. The default is False. This feature is not supported when the *mode* parameter value is 'mol3D'.

Returns

plot
Returns a plotted graph.

Functions

Function Name	Description
get_node_positions	Finds the (x,y)-coordinates of nodes on the basis of the specified layout and seed values.

get_node_positions

Finds the (x,y)-coordinates of nodes on the basis of the specified layout and seed values.

API: get_node_positions

```
sasviya.network.utils.visualization.get_node_positions(g,
layout='kamada_kawai', seed=None, scale_factor=None, **kwargs)
```

Parameters

g : SAS graph object
Specifies the graph to plot.

layout : str
Specifies the algorithm to compute the node positions. The default is 'kamada_kawai'. The following layouts are supported: 'bipartite', 'circo', 'circular', 'dot', 'fdp', 'fruchterman_reingold', 'kamada_kawai', 'multipartite', 'neato', 'planar', 'random', 'sfdp', 'shell', 'spectral', 'spiral', 'spring', and 'twopi'. The 'circo', 'dot', 'fdp', 'neato', 'sfdp', and 'twopi' layouts require Graphviz and pydot to be installed.

seed : int
Specifies the seed for reproducibility in random layouts.

scale_factor : float, optional
Specifies the scale factor for positions.

Returns

dict of arrays
Returns the x- and y-coordinates of each node.

30

Utilities

Functions

Function Name	Description
apply	Applies a function to all node or edge attributes and updates them.
contract_edges	Returns the induced graph by subset of edges and removes nodes that are not inside the edges subset.
copy	Returns a copy of the input graph.
drop_edge_attributes	Removes edge attributes if they exist.
drop_node_attributes	Removes node attributes if they exist.
edge_subgraph	Returns the induced graph by subset of edges.
get_all_edges_attributes	Returns a set of of edge attributes.
get_all_nodes_attributes	Returns a set of of node attributes.
get_colormap_from_rgb_list	Converts a list of RGB colors to a colormap.
get_edge_attributes	Returns the value of the edge attribute that is specified by the *name* parameter as a dictionary.
get_meta_graph	Constructs a metagraph from the input graph.
get_node_attributes	Returns the value of the node attribute that is specified by the *name* parameter as a dictionary.
has_edge_attribute	Checks the edges of the input graph for the specified edge attribute.

Function Name	Description
has_node_attribute	Checks the nodes of the input graph for the specified node attribute.
has_selfloops	Determines whether the input graph has any self-loops.
induced_subgraph	Returns a subgraph of g that contains only the nodes that are listed in the nodes_subset parameter.
is_bipartite	Determines whether the input graph is bipartite.
is_cycle	Determines whether the input graph is a simple cycle.
is_dag	Determines whether the input graph is a directed acyclic graph (DAG).
is_path	Determines whether the input graph is a simple path.
is_solution_null	Checks whether the solution is null.
remove_isolated_nodes	Removes isolated nodes from a graph.
remove_selfloops	Removes all self-loops from the input graph.
rename_edge_attributes	Renames edge attributes.
rename_node_attributes	Renames node attributes.
reverse	Returns a copy of a directed graph with reversed edges.
set_edge_attributes	Sets edge attributes by using a given value or dictionary of values.
set_node_attributes	Sets node attributes from a given value or dictionary of values.
simplify	Computes a simple graph by consolidating multiedges.
split	Splits input data into non-overlapping data sets.
subdivision_graph	Creates a new graph by inserting a node for each edge in the input graph.
subgraph	Returns a subgraph of g, as determined by the nodes_subset and edges_subset parameter values.
total_wt_by_edge_attr	Calculates the total sum of the specified edge weight attributes in the graph.

apply

Applies a function to all node or edge attributes and updates them.

API: apply

```
sasviya.network.utils.utils.apply(g, node_attr_handler=None,
edge_attr_handler=None, immutable=None, *, inplace=False)
```

The function receives node or edge attributes as a dictionary and return a dictionary to be used to update the attributes.

Parameters

g : SAS graph object
Specifies the input graph to assess.

node_attr_handler : function or None, optional
Specifies the function to apply to node attributes. The default is None.

edge_attr_handler : function or None, optional
Specifies the function to apply to edge attributes. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the input graph. The default is False.

Returns

SAS graph object or None.
Returns the resulting graph after the function is applied, or returns nothing if the *inplace* parameter value is True.

Examples

```
import sasviya.network as sasnet
G = sasnet.Graph([[1,2,{}],[1,3,{'w':1}],[1,4,{'w':2}]])
sasnet.set_node_attributes(G,{1:10,2:20,3:30,4:40},name='x',inplace=True)
```

#use apply to fillna >>> sasnet.apply(G, edge_attr_handler=lambda data:{‘w’:0 if ‘w’ not in data else data[‘w’]},inplace=True) >>> assert G.edges[(1,2)][‘w’] == 0 >>> print(G.nodes(data=True)) #use apply to negate an attribute >>> sasnet.apply(G, node_attr_handler=lambda data:{‘x’: -data[‘x’]},inplace=True) >>> assert G.nodes[1][‘x’]== -10

contract_edges

Returns the induced graph by subset of edges and removes nodes that are not inside the edges subset.

API: contract_edges

sasviya.network.utils.utils.contract_edges(g, edges_subset, how=None, immutable=None, _contracted_node_map=None, _contracted_edge_map=None, *, inplace=False, aggregate=False, selfloops=False)

Parameters

g : SAS graph object
Specifies the input graph to assess.

edges_subset : list (or any container or Pandas DataFrame) of edges
Specifies the subset of edges in the form of two-tuples [(u,v)] or three-tuples [(u,v,edge_key)]. For example, [(1,2),(3,4)] or [(1,2,0),(1,2,1),(2,1,0)].

how : function or str or dict or None, optional
Specifies how to aggregate or select edge attributes. The default is None. When the *aggregate* parameter value is False (the default) or an empty dictionary, the graph is simplified by removing multiedges and all edge attributes. When the *aggregate* parameter value is True, the *how* parameter value can be a function such as *np.mean* or *np.sum*; a string such as ‘min’, ‘max’, or ‘mean’; or a dictionary

that specifies different aggregation methods for different attributes. None is a special key that is used for all attributes not explicitly specified in that dictionary.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether to remove self-loops from the input graph directly or to create a copy of the graph with no self-loops. The default is False.

aggregate : bool, optional
Specifies whether to aggregate or select the multiedge attributes. The default is False, which means that the output potentially contains multiedges. When the value is True, the multiedge attributes are aggregated using the method that is specified in the *how* parameter. When the value is False, an edge is selected from each multiedge according to the method that is specified in the *how* parameter.

selfloops : bool, optional
Specifies whether or not to keep self-loops. The default is False.

Returns

SAS graph object
Returns a graph that is generated by performing edge contraction and then consolidating mutliedges of a multigraph. If the *selfloops* parameter value is False, the output graph is a simple graph; if the value is True, it is a multigraph with no multiedges.

copy

Returns a copy of the input graph.

API: copy

```
sasviya.network.utils.utils.copy(g, immutable=None, *, data=True)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, meaning that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns an independent copy of the input.

drop_edge_attributes

Removes edge attributes if they exist.

API: drop_edge_attributes

```
sasviya.network.utils.utils.drop_edge_attributes(g, attributes_subset=None,
edges_subset=None, immutable=None, *, inplace=False)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

attributes_subset : str or list, optional
Specifies the edge attribute or a list of the edge attributes to remove; for example, 'weight' or ['weight','color']. By default, all edge attributes are removed.

edges_subset : list or iterable of edges, optional
Specifies the edges to remove attributes from; for example, [(1,2),(2,3)]. By default the attributes for all edges are removed.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by removing the specified edge attributes. If the *inplace* parameter value is False, a copy of the input graph with the modified edge attributes is returned.

drop_node_attributes

Removes node attributes if they exist.

API: drop_node_attributes

```
sasviya.network.utils.utils.drop_node_attributes(g, attributes_subset=None,
nodes_subset=None, immutable=None, *, inplace=False)
```

Parameters

g : SAS graph object
Specifies the input graph to remove node attributes from.

attributes_subset : str or list, optional
Specifies the node attribute or list of node attributes to remove; for example, ['weight','color']. By default, all node attributes are removed.

nodes_subset : list or iterable of nodes, optional
Specifies the nodes to drop attributes from; for example, [1,2,3] or (1,2,3) or [1] or (1,). By default, the attributes for all nodes are removed.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by removing the specified node attributes. If the *inplace* parameter value is False, a copy of the input graph with the modified node attributes is returned.

edge_subgraph

Returns the induced graph by subset of edges.

API: edge_subgraph

```
sasviya.network.utils.utils.edge_subgraph(g, edges_subset, immutable=None,
_small_set_optimized=None, *, data=True, keep_all_nodes=False)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

edges_subset : list (or any container or Pandas DataFrame) of edges
Specifies the subset of edges in the form of two-tuples [(u,v)] or three-tuples [(u,v,edge_key)]. For example, [(1,2),(3,4)] or [(1,2,0),(1,2,1),(2,1,0)].

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, meaning that the node and edge attributes of the graph are retained.

keep_all_nodes : bool, optional
Specifies whether or not to keep the nodes that are not inside the edges subset. The default is False.

Returns

SAS graph object
Returns the edge-induced subgraph.

get_all_edges_attributes

Returns a set of of edge attributes.

API: get_all_edges_attributes

```
sasviya.network.utils.utils.get_all_edges_attributes(g)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

Returns

set
Returns a set of edge attributes.

get_all_nodes_attributes

Returns a set of of node attributes.

API: get_all_nodes_attributes

```
sasviya.network.utils.utils.get_all_nodes_attributes(g)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

Returns

set
Returns a set of node attributes.

get_colormap_from_rgb_list

Converts a list of RGB colors to a colormap.

API: get_colormap_from_rgb_list

```
sasviya.network.utils.color.get_colormap_from_rgb_list(rgb_list)
```

Parameters

rgb_list : list of list of int
Specifies the input list of lists. Each element should be a length-3 list of red, green, and blue values, respectively. The values are between 0 and 255.

Returns

ListedColormap
Returns the colormap equivalent.

get_edge_attributes

Returns the value of the edge attribute that is specified by the *name* parameter as a dictionary.

API: get_edge_attributes

```
sasviya.network.utils.utils.get_edge_attributes(g, name)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

name : str
Specifies the name of the edge attribute.

Returns

dict
Returns a dictionary in which the keys are the edges and the values are the attribute values.

get_meta_graph

Constructs a metagraph from the input graph.

API: get_meta_graph

```
sasviya.network.utils.utils.get_meta_graph(g, node_type, edge_type=None,
immutable=None)
```

Each node in the metagraph corresponds to a unique type of node in the input graph, and each edge in the metagraph represents a unique type of connection between these node types in the input graph.

Parameters

g : SAS graph object
Specifies the input graph to assess.

node_type : str
Specifies the node attribute to consider in constructing the metagraph.

edge_type : str, optional
Specifies the edge attribute to consider in constructing the metagraph. By default, all connections between nodes are considered the same.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

Returns

SAS graph object
Returns the metagraph, which has an edge attribute count and determines the number of unique connections between pairs of nodes.

get_node_attributes

Returns the value of the node attribute that is specified by the *name* parameter as a dictionary.

API: get_node_attributes

```
sasviya.network.utils.utils.get_node_attributes(g, attr)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

attr : str
Specifies the name of the node attribute.

Returns

dict
Returns a dictionary in which the keys are the nodes and the values are the attribute values.

has_edge_attribute

Checks the edges of the input graph for the specified edge attribute.

API: has_edge_attribute

```
sasviya.network.utils.utils.has_edge_attribute(g, name, *, all_edges=False)
```

Determines whether an edge attribute that is specified by the *name* parameter exists in all or any edges in the input graph.

Parameters

g : SAS graph object
Specifies the input graph to assess.

name : str
Specifies the name of the edge attribute.

all_edges : bool, optional
Specifies whether or not to check for the existence of the edge attribute in all or any of the edges. The default is False.

Returns

bool
If the all_edges parameter value is True, the value True is returned if that edge attribute exists in all edges; otherwise the value False is returned. If the all_edges parameter value is False, the value True is returned if at least one edge in the graph has that specified edge attribute.

has_node_attribute

Checks the nodes of the input graph for the specified node attribute.

API: has_node_attribute

```
sasviya.network.utils.utils.has_node_attribute(g, name, *, all_nodes=False)
```

Determines whether a node attribute that is specified by the *name* parameter exists in all or any nodes in the input graph.

Parameters

g : SAS graph object
Specifies the input graph to assess.

name : str
Specifies the name of the node attribute.

all_nodes : bool, optional
Specifies whether or not to check for the existence of the node attribute in all or any of the nodes. The default is False.

Returns

bool
If the all_nodes parameter value is True, the value True is returned if that node attribute exists in all node; otherwise the value False is returned. If the all_nodes parameter value is False, the value True is returned if at least one node in the graph has that specified node attribute.

has_selfloops

Determines whether the input graph has any self-loops.

API: has_selfloops

```
sasviya.network.utils.utils.has_selfloops(g)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

Returns

bool
Returns a value of True if the graph contain any self-loops; otherwise a value of False is returned.

induced_subgraph

Returns a subgraph of g that contains only the nodes that are listed in the nodes_subset parameter.

API: induced_subgraph

```
sasviya.network.utils.utils.induced_subgraph(g, nodes_subset,
immutable=None, *, data=True)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

nodes_subset : list (or any container) of nodes
Specifies the subset of nodes.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, meaning that the node and edge attributes of the graph are retained.

Returns

SAS graph object
Returns the induced subgraph.

is_bipartite

Determines whether the input graph is bipartite.

API: is_bipartite

```
sasviya.network.utils.utils.is_bipartite(g)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

Returns

bool
Indicates whether the input graph is bipartite.

is_cycle

Determines whether the input graph is a simple cycle.

API: is_cycle

```
sasviya.network.utils.utils.is_cycle(g)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

Returns

bool
Indicates whether the input graph is a simple cycle (True) or not (False).

is_dag

Determines whether the input graph is a directed acyclic graph (DAG).

API: is_dag

```
sasviya.network.utils.utils.is_dag(g)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

Returns

bool
Indicates whether the input graph is a directed acyclic graph (DAG).

is_path

Determines whether the input graph is a simple path.

API: is_path

```
sasviya.network.utils.utils.is_path(g)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

Returns

bool
Indicates whether the input graph is a simple path (True) or not (False).

is_solution_null

Checks whether the solution is null.

API: is_solution_null

```
sasviya.network.utils.utils.is_solution_null(solution)
```

An algorithm solution is null when the problem is infeasible.

Parameters

solution : object
Specifies the solution that is obtained from an algorithm.

Returns

bool
Returns a value of True if the solution is null.

remove_isolated_nodes

Removes isolated nodes from a graph.

API: remove_isolated_nodes

```
sasviya.network.utils.utils.remove_isolated_nodes(g, immutable=None, *,
inplace=False)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the input graph. The default is False.

Returns

SAS graph object or None
Returns the resulting graph after isolated nodes are removed, or returns nothing if the *inplace* parameter value is True.

remove_selfloops

Removes all self-loops from the input graph.

API: remove_selfloops

```
sasviya.network.utils.utils.remove_selfloops(g, *, inplace=False)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

inplace : bool, optional
Specifies whether to remove self-loops from the input graph directly or to create a copy of the graph with no self-loops. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is False, a copy of the input graph with no self-loops is returned; otherwise nothing is returned.

rename_edge_attributes

Renames edge attributes.

API: rename_edge_attributes

```
sasviya.network.utils.utils.rename_edge_attributes(g, rename_dict,
immutable=None, *, inplace=False)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

rename_dict : dict
Specifies a dictionary of edge attributes in which the keys are the old names and the values are the new names.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by renaming the specified edge attributes. If the *inplace* parameter value is False, a copy of the input graph with renamed edge attributes is returned.

rename_node_attributes

Renames node attributes.

API: rename_node_attributes

```
sasviya.network.utils.utils.rename_node_attributes(g, rename_dict,
immutable=None, *, inplace=False)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

rename_dict : dict
Specifies a dictionary of node attributes in which the keys are the old names and the values are the new names.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by renaming the specified node attributes. If the *inplace* parameter value is False, a copy of the input graph with renamed node attributes is returned.

reverse

Returns a copy of a directed graph with reversed edges.

API: reverse

```
sasviya.network.utils.utils.reverse(g, *, data=True)
```

Parameters

g : SAS directed graph object
Specifies the input directed graph to reverse.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, meaning that the node and edge attributes of the graph are retained.

Returns

directed graph object
Returns a directed graph that holds the reversed edges.

set_edge_attributes

Sets edge attributes by using a given value or dictionary of values.

API: set_edge_attributes

sasviya.network.utils.utils.set_edge_attributes(g, values, name=None, immutable=None, *, inplace=False)

Parameters

g : SAS graph object
Specifies the input graph to assess.

values : scalar or dict-like or Pandas DataFrame
Specifies what the edge attribute should be set to. When this parameter is set to a Pandas DataFrame, it is expected to have from, to, and edge_key (only for multigraphs) columns, and the other columns are treated as edge attributes to add. When *values* is a dictionary, the keys are edges and the values are either edge attribute values or dictionaries of attribute name-value pairs. For multigraphs, the edge tuples must be of the form (u, v, key). For simple graphs, the keys must be tuples of the form (u, v). When this parameter is not set to a dictionary, it is treated as a single attribute value that is then applied to every edge in *g*. The attribute name is specified by the *name* parameter.

name : str or None, optional
Specifies the name of the edge attribute to set if the *values* parameter is set to a scalar. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not to modify the graph in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is False, a copy of the input graph with the new edge attributes is returned. If the *inplace* parameter value is True, the graph is modified by setting the specified edge attributes.

set_node_attributes

Sets node attributes from a given value or dictionary of values.

API: set_node_attributes

```
sasviya.network.utils.utils.set_node_attributes(g, values, name=None,
immutable=None, *, inplace=False)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

values : scalar or dict-like or Pandas DataFrame
Specifies what the node attribute should be set to. When *values* is not a dictionary, it is treated as a single attribute value that is then applied to every node in *g*. This means that if you provide a mutable object, such as a list, updates to that object are reflected in the node attribute for every node. The attribute name is specified by the *name* parameter. When *values* is a dictionary, the keys are nodes and the values are either node attribute values or dictionaries of attribute name-value pairs.

name : str or None, optional
Specifies the name of the node attribute to set if the *values* parameter is set to a scalar. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

inplace : bool, optional
Specifies whether or not the graph is modified in place. The default is False.

Returns

SAS graph object or None
If the *inplace* parameter value is True, the graph is modified by setting the specified node attributes. If the *inplace* parameter value is False, a copy of the input graph with the new node attributes is returned.

simplify

Computes a simple graph by consolidating multiedges.

API: simplify

```
sasviya.network.utils.utils.simplify(g, how=None, immutable=None, *,
aggregate=True, selfloops=False)
```

It does so by either aggregating or selecting the edges within the input multigraph.

Parameters

g : SAS graph object
Specifies the input multiedges graph to assess.

how : function or str or dict or None, optional
Specifies how to aggregate or select edge attributes. The default is None. When the value is None or an empty dictionary, the graph is simplified by removing multiedges and all edge attributes. When the *aggregate* parameter value is True, the *how* parameter value can be a function such as np.mean or np.sum; a string such as 'first', 'last', or 'sum'; or a dictionary that specifies different aggregation methods for different attributes. None can be used as a special key for all attributes that are not explicitly specified in that dictionary. When the *aggregate* parameter value is False, the *how* parameter value can be a string such as 'first' or 'last' or an ordered dictionary with the edge attributes as its key and the selection method as its value. For example, {'weight':'min','type':'max'}. The keys of the dictionary are used in the specified order to select an edge among multiedges. If ties still occur, then by default, the edge with the smallest edge key value is selected. To overwrite this behavior, the *edge_key* parameter should be added to the specified dictionary with 'max' as its value.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

aggregate : bool, optional
Specifies whether or not to aggregate or select the multiedge attributes. The default is True, which means that the multiedge attributes are aggregated by using the method that is specified in the *how* parameter. When the value is False, an edge is selected from each multiedges according to the method that is specified in the *how* parameter.

selfloops : bool, optional
Specifies whether or not to keep self-loops. The default is False.

Returns

SAS graph object
Returns a graph that is generated by consolidating mutliedges of a multigraph. If the *selfloops* parameter value is False, the output graph is a simple graph; if the value is True, it is a multigraph with no multiedges.

split

Splits input data into non-overlapping data sets.

API: split

```
sasviya.network.utils.utils.split(data, sizes, seed=None, group_by=None,
event=None, event_prop=None)
```

Parameters

data : Pandas DataFrame or Pandas Series or SAS Table name
Specifies the data to be split.

sizes : number or list of numbers
Specifies samples sizes to use in partitioning data. When the value is float, it should be in the range (0.0, 1.0] and represent the proportion of the data set to include in the first split. The second split contains the rest of the data set. When

the value is int, it should be in the range (0,len(data)] and represent the absolute number of samples in the first split. The second split contains the rest of the data set. When the value is a list of floats, each item should be in the range (0.0, 1.0] and their sum should be less than or equal to 1.0. When the value is a list of ints, each item should be in the range (0,len(data)] and their sum should be less than or equal to len(data).

seed : int or None, optional
Specifies a random seed for reproducibility. The default is None.

group_by : str or list of str or None, optional
Specifies the name of variable or variables to use for grouping results. The default is None.

event : str or None, optional
Specifies the rare event level for oversampling. The default is None.

event_prop : float or None, optional
Specifies the proportion of rare events in the sample for oversampling. The default is None.

Returns

tuple
Returns a tuple that contains partitioned Pandas DataFrames.

subdivision_graph

Creates a new graph by inserting a node for each edge in the input graph.

API: subdivision_graph

```
sasviya.network.utils.utils.subdivision_graph(g, edge_subset=None,
from_attr=None, to_attr=None, edge_key_attr=None, immutable=None, *,
data=True)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

edge_subset : collection of tuples or None
Specifies the edges to subdivide. The default is None, meaning that all edges are subdivided.

from_attr : str or None, optional
Specifies the node attribute to store the value of an edge's *from* node in the original graph. The default is None.

to_attr : str or None, optional
Specifies the node attribute to store the value of an edge's *to* node of the original graph. The default is None.

edge_key_attr : str or None, optional
Specifies the node attribute to store the edge key of the original graph. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

data : bool, optional
Specifies whether or not to retain attributes. When the value is False, these attributes are deleted. The default is True, which means that the edge attributes are transferred to the nodes that are created for each edge.

Returns

SAS graph object
Returns the resulting graph after subdivision.

subgraph

Returns a subgraph of g, as determined by the nodes_subset and edges_subset parameter values.

API: subgraph

```
sasviya.network.utils.utils.subgraph(g, nodes_subset=None,
edges_subset=None, immutable=None, *, data=True, induced=False)
```

Parameters

g : SAS graph object
Specifies the input graph to assess.

nodes_subset : iterable of nodes or None, optional
Specifies the subset of nodes to include in the subgraph. The default is None.

edges_subset : iterable of edges or None
Specifies the subset of edges in the form of two-tuples [(u,v)] or three-tuples [(u,v,edge_key)]; for example, [(1,2),(3,4)] or [(1,2,0),(1,2,1),(2,1,0)]. The default is None.

immutable : bool or None, optional
Specifies whether or not to return an immutable copy. The default is None, which means that the mutability matches the input graph.

data : bool, optional
Specifies whether or not to retain attributes. The default is True, meaning that the node and edge attributes of the graph are retained.

induced : bool, optional
Specifies whether or not to call the induced_subgraph function and to add nodes that are listed in the *nodes_subset* parameter as well as the edges incident on those nodes. The default is False, which means that only the list of nodes is added to the output graph.

Returns

SAS graph object
Returns a subgraph of the graph, as determined by the different values of the nodes_subset and edges_subset parameters. When the nodes_subset parameter is specified but the edges_subset parameter value is None: If the induced parameter value is True, the output graph contains all nodes that are listed in the nodes_subset parameter and all edges incident on those nodes. If the induced parameter value is False, the output graph contains only the nodes that are listed in the nodes_subset parameter; no edges are added. When the edges_subset parameter is specified but the nodes_subset parameter value is None: The output graph contains all edges that are listed in the edges_subset parameter and all nodes (from,to) that are listed in the edges_subset parameter. When both the edges_subset and nodes_subset parameters are specified: If the induced parameter value is True, the output graph includes all nodes that are

listed in either the edges_subset or nodes_subset parameter as well as all edges incident to those nodes. If the induced parameter value is False, the output graph contains any node that is listed in the nodes_subset or edges_subset parameter, but it contains only edges that are listed in the edges_subset parameter.

total_wt_by_edge_attr

Calculates the total sum of the specified edge weight attributes in the graph.

API: total_wt_by_edge_attr

```
sasviya.network.utils.utils.total_wt_by_edge_attr(g, weight_attr=None)
```

Parameters

g : SAS graph object
Specifies the graph for which edge weights are summed.

weight_attr : str or None, optional
Specifies the edge attributes to sum. The default is None, meaning that unit weights are used.

Returns

int or float
Returns the sum of specified attributes over all edges in the graph.

1

Examples

Community Detection Algorithms

Community Detection on Zachary's Karate Club Data

This example uses Zachary's Karate Club data (Zachary 1977), which describe social network friendships among 34 members of a karate club at a university in the United States in the 1970s. This is one of the standard publicly available data sets that analysts use to test community detection algorithms. It contains 34 nodes and 78 edges.

To read more about the community detection algorithms, refer to SAS Viya Platform Documentation.

Importing the library

```
# Import libraries
import sasviya.network as sasnet
import matplotlib.pyplot as plt
```

Creating the graph

The Karate Club graph is available as a built-in example in the `built_in_graphs` module from the `sasviya.network` package. This makes it easy to load and experiment with standard graphs commonly used in network analysis.

To explore other built-in graphs, visit Built-In Graph Utilities.

```
# Creating the graph
G = sasnet.karate_club_graph()
```

Running community detection

You can apply the Louvain algorithm to the graph for community detection as follows. You use different resolution values to observe how the number and size of detected communities are affected.

```
# Running Community detection
def run_community(G, res_level):
    outG = G.louvain(resolution=res_level)
    print(f'Communities (res={res_level})')
    for i, comm in enumerate(outG):
        print(f'comm {i+1}: {comm.nodes()}')

run_community(G, 1)
```

```
Communities (res=1)
comm 1: [0, 1, 2, 3, 7, 9, 11, 12, 13, 17, 19, 21]
comm 2: [4, 5, 6, 10, 16]
comm 3: [8, 14, 15, 18, 20, 22, 26, 29, 30, 32, 33]
comm 4: [23, 24, 25, 27, 28, 31]
```

```
run_community(G, 0.5)
```

```
Communities (res=0.5)
comm 1: [0, 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 16, 17, 19, 21]
comm 2: [8, 14, 15, 18, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33]
```

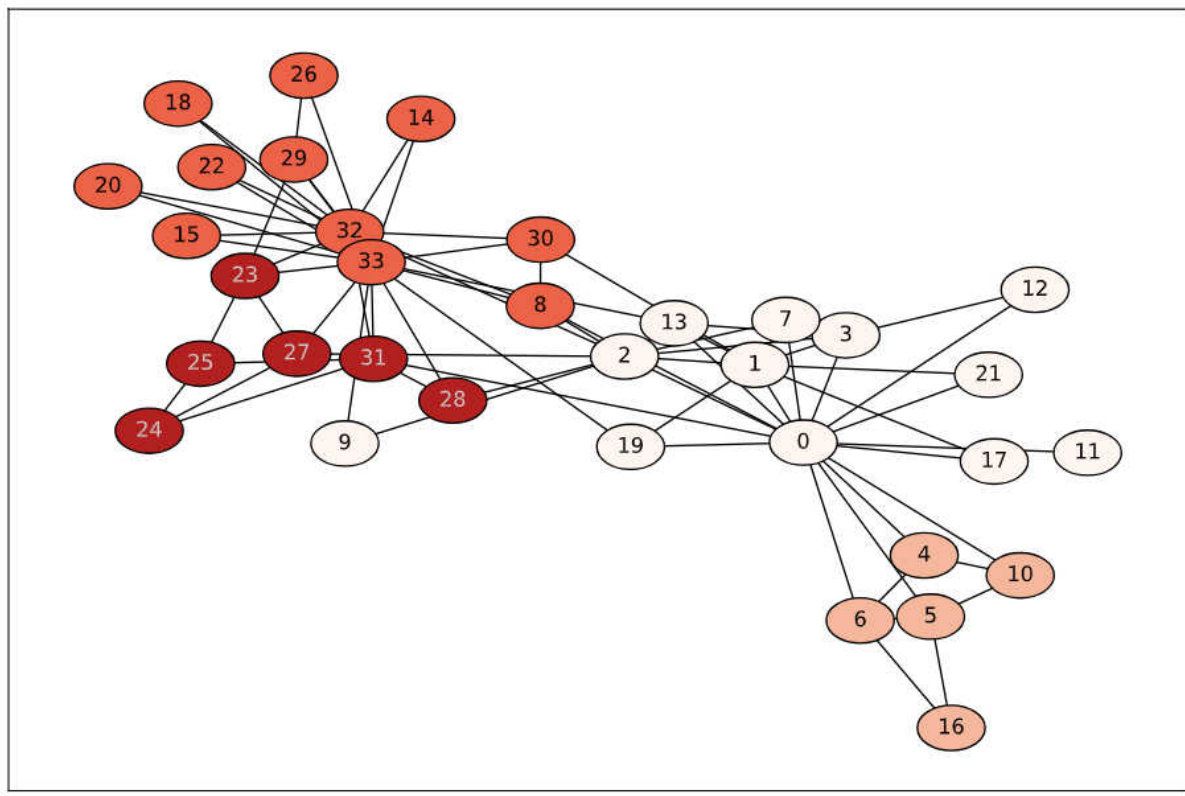
Visualizing Louvain Community Detection

You use the `louvain_node` function as follows to detect communities and to store the community number as a node attribute, enabling easier visualization:

```
# Visualizing Louvain Community Detection
sasnet.louvain_nodes(G, resolution=1, inplace=True)
print(G.to_pandas_nodes().head())
```

	node	club	community
0	0	Mr. Hi	0
1	1	Mr. Hi	0
2	2	Mr. Hi	0
3	3	Mr. Hi	0
4	4	Mr. Hi	1

```
fig, ax = plt.subplots(figsize=(12,8))
sasnet.PlotGraph(G, layout='fruchterman_reingold',
nodes_color_by_value='community').show(ax=ax)
SAS.pyplot(fig)
```



Reference

Zachary, W. W. (1977). “An Information Flow Model for Conflict and Fission in Small Groups.” *Journal of Anthropological Research* 33:452–473.

Core Algorithms

Calculate the Core Decomposition of a Graph

This notebook provides an example of building a graph of English words to demonstrate the use of the core decomposition algorithm. The idea for this example is from Batagelj and Zaversnik (2011).

You use the Python library wordfreq to obtain the 5,000 most frequently used English words. Then you construct a graph by connecting words if one word can be transformed into the other word by inserting, removing, or changing one letter. This step requires calculating the edit distance by using the edit_distance package. Ultimately, you use the `core_decomposition` and `add_core_number_attr functions` to identify the cores of the graph.

In a graph, $G = (N, E)$, an induced maximal subgraph $G_s = (S, E_s)$ is a k-core if and only if every node in $G_s$has a degree greater than or equal to k. To learn more about the `core_decomposition` function, see the SAS Viya Platform Documentation.

Importing the libraries

```
# Install required packages
import subprocess
import sys
import os
sys.path.append('/tmp')
subprocess.check_call([sys.executable, "-m", "pip", "install",
"editdistance","wordfreq",
                       "--target","/tmp","--upgrade"])

# Import libraries
import sasviya.network as sasnet
from itertools import combinations
import matplotlib.pyplot as plt
import editdistance
from wordfreq import top_n_list
```

Extracting frequent english words and filtering

First you obtain a list of 5,000 frequently used English words by using the wordfreq package as follows. To refine the list, you exclude words that consist of a single

character or words that end in *'s* in order to produce a more suitable set of common English words for the downstream analysis.

```
# Extracting frequent english words and filtering
common_words = top_n_list('en', 5000)
words = [word for word in common_words if len(word)>1 and "'s" not in
word]
print(f'number of words in the list: {len(words)}')
print(f'first 10 words: {words[:10]}')
```

```
number of words in the list: 4936
first 10 words: ['the', 'to', 'and', 'of', 'in', 'is', 'for', 'that', 'you', 'it']
```

Constructing graph of words using edit distance

The following code generates all possible pairs of words and computes the edit distance for each pair. Only the pairs whose edit distance is equal to 1 are retained. This means that you can transform one word into the other by inserting, deleting, or replacing a single character. Based on these pairs, a graph is constructed in which each word is represented as a node, and an edge is added between nodes that differ by exactly one edit.

```
# Constructing graph of words using edit distance
edges = [(a,b) for a,b in combinations(words, 2) if
editdistance.eval(a, b) == 1]

G = sasnet.Graph(edges, nodes_data=words)
print(G)
print(G.to_pandas_edges().head())
```

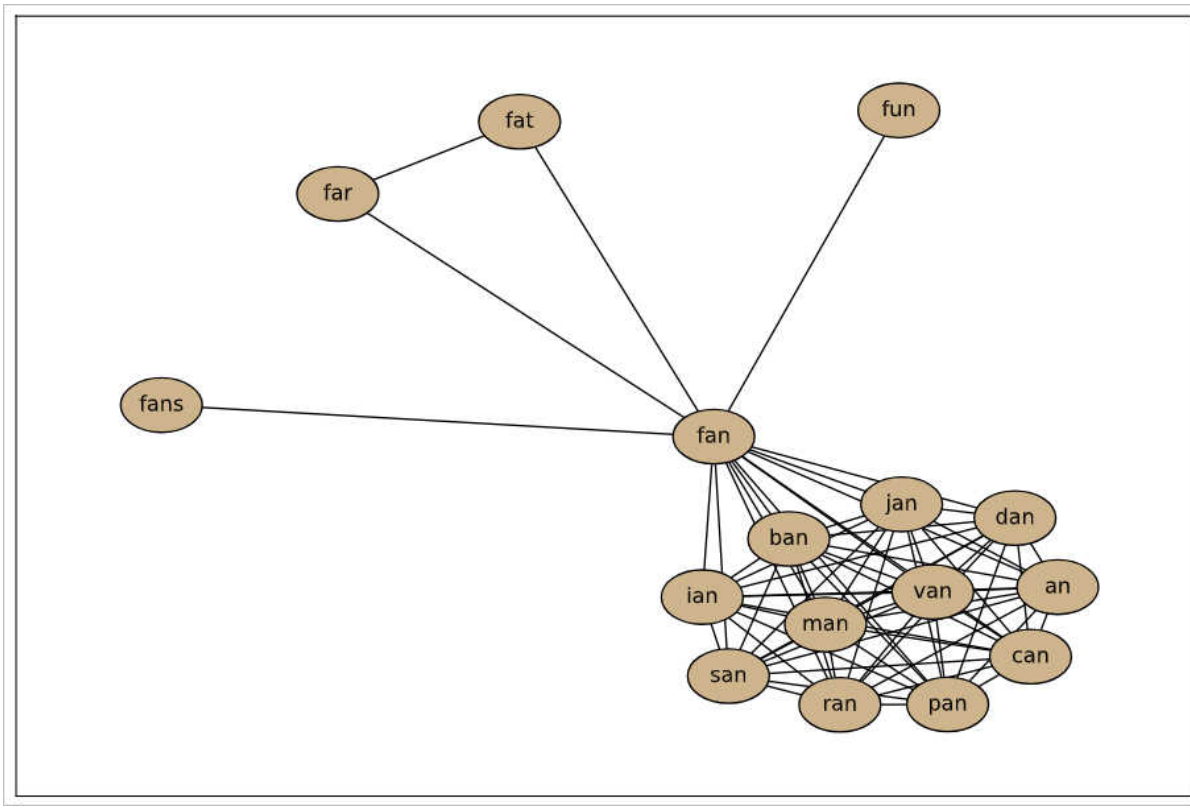
```
Graph object with 4936 nodes and 5042 edges
  from    to
0  the    he
1  the  they
2  the   she
3  the  them
4  the  then
```

Visualizing the local neighborhood of a particular word

To explore the local structure around a specific word, `fan`, you first identify a subset of nodes. This subset includes the word `fan` and all words in the graph that are directly connected to it.

You then extract an induced subgraph by using this subset of nodes as in the following statements; all edges among them are preserved. You produce a visualization of the resulting subgraph to provide insight into the immediate connections and relationships that surround the target word.

```
# Visualizing the local neighborhood of a particular word
fig, ax = plt.subplots(figsize=(12,8))
nodes_subset = list(G.neighbors('fan')) + ['fan']
induced_fan  = sasnet.induced_subgraph(G, nodes_subset=nodes_subset)
sasnet.PlotGraph(induced_fan, layout='fruchterman_reingold',
nodes_color='tan').show(ax=ax)
SAS.pyplot(fig)
```



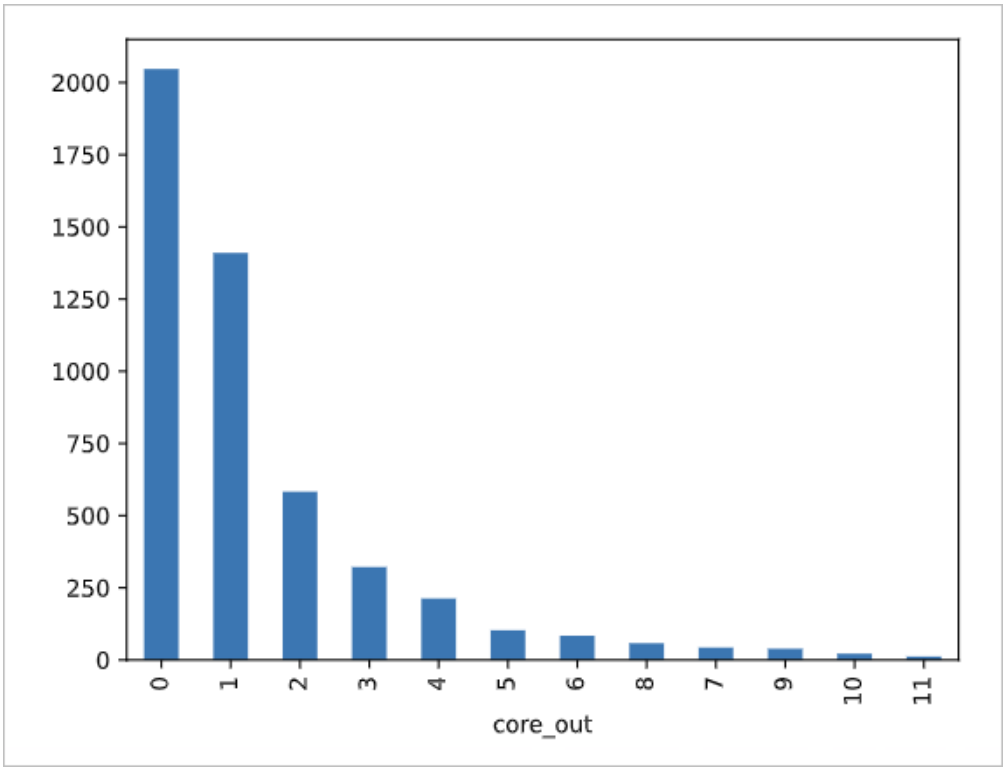
Analyzing graph structure using core numbers

You compute core numbers and assign them to each node in the graph as follows by using the `add_core_number_attr` function, which stores the core value as a node attribute. You then examine the distribution of nodes across different core values by plotting the number of nodes that are associated with each core level. This helps identify the density and hierarchy of connectivity in the graph.

```
# Analyzing graph structure using core numbers
G.add_core_number_attr(inplace=True)
nodes_df = G.to_pandas_nodes()
print(nodes_df.head())
```

	node	core_out
0	the	4
1	he	9
2	they	3
3	she	4
4	them	3

```
fig, ax = plt.subplots()
nodes_df['core_out'].value_counts().plot.bar(ax=ax)
SAS.pyplot(fig)
```



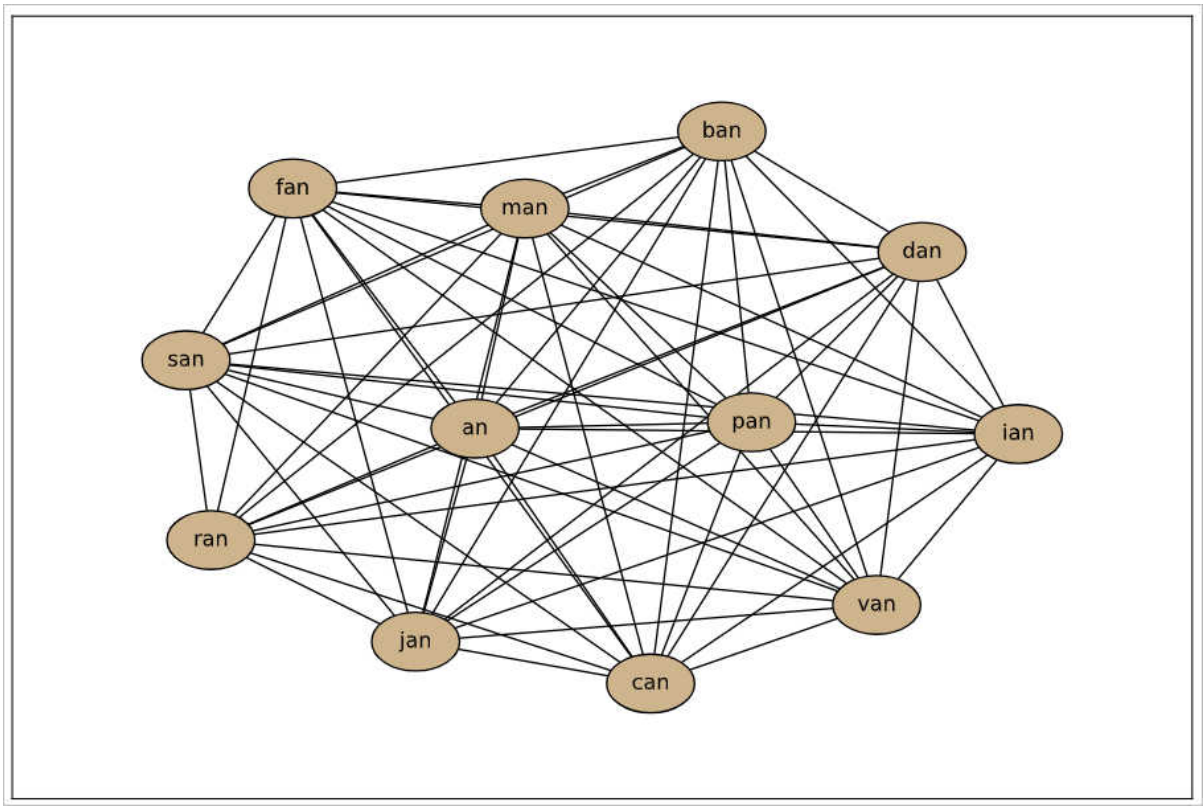
Visualizing the densest core of the graph

You use the `core_decomposition` function as follows to compute the core structure of the graph. This decomposition reveals nested layers of connectivity that are based on node degree within increasingly dense subgraphs. You then extract and visualize the subgraph that corresponds to the highest core value.

```
# Visualizing the densest core of the graph
cores = list(G.core_decomposition())
G_11_core = cores[11]
print(G_11_core)
```

```
Graph object with 12 nodes and 66 edges
```

```
fig, ax = plt.subplots(figsize=(12,8))
sasnet.PlotGraph(G_11_core, layout='fruchterman_reingold',
nodes_color='tan').show(ax=ax)
SAS.pyplot(fig)
```



Reference

Batagelj, V., and Zavernsik, M. (2011). "An O(m) Algorithm for Cores Decomposition of Networks." *Advances in Data Analysis and Classification* 5: 129–145.

Projection Algorithms

Using Projection to Recommend Recipe Ingredient Pairings

This example addresses a recommendation problem by linking similar ingredients together on the basis of recipe data. It uses projection algorithms to identify pairs of ingredients that appear together in one or more recipes. The input data consist of six recipes, and each recipe lists a set of ingredients. Through this approach, you uncover relationships between ingredients, gaining insight into common pairings that can support recommendation or discovery tasks.

To learn more, visit the SAS Viya Platform Documentation.

Importing the library

```
# Import libraries
import sasviya.network as sasnet
import matplotlib.pyplot as plt
```

Defining the input graph

The following input data are organized as a dictionary in which each recipe is associated with a list of its ingredients. This dictionary is treated as an adjacency list to form a bipartite network, which represents the relationship between recipes and their ingredients.

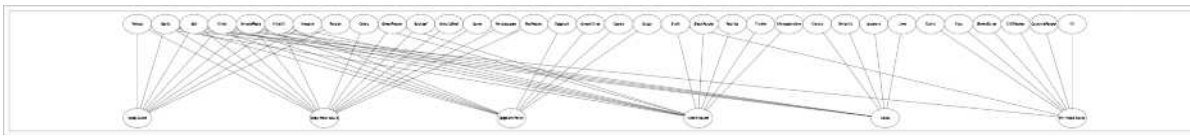
```
# Define the input graph
adj = {
         'Spag Sauce':['Tomato', 'Garlic', 'Salt', 'Onion',
'TomatoPaste', 'OliveOil',
                       'Oregano', 'Parsley'],
    'Spag Meat Sauce':['Tomato', 'Garlic', 'Salt', 'Onion',
'TomatoPaste', 'OliveOil',
                       'Celery', 'GreenPepper', 'BayLeaf',
'GroundBeef', 'Carrot',
                       'PorkSausage', 'RedPepper'],
    'Eggplant Relish':['Garlic', 'Salt', 'Onion', 'TomatoPaste',
'OliveOil', 'Eggplant',
                       'GreenOlives', 'Capers', 'Sugar'],
       'Creole Sauce':['Tomato', 'Salt', 'Onion', 'TomatoPaste',
'OliveOil', 'Celery',
                       'Broth', 'GreenPepper', 'BlackPepper',
'Paprika', 'Thyme',
                       'Worcestershire'],
             'Salsa':['Tomato', 'Garlic', 'Salt', 'Onion',
'Cilantro', 'Tomatillo',
                       'Jalapeno', 'Lime'],
    'Enchilada Sauce':['Tomato', 'Broth', 'Cumin', 'Flour',
'BrownSugar', 'ChiliPowder',
                       'CayennePepper', 'Oil']
}
G = sasnet.Graph(adj)
print(G)
```

```
Graph object with 40 nodes and 58 edges
```

Displaying the input graph

```
# Display the input graph
```

```
bipart_node_pos = sasnet.visualization.get_node_positions(G,
layout='bipartite', nodes=adj.keys(), align="horizontal")
fig, ax = plt.subplots(figsize=(80,8))
sasnet.PlotGraph(G, nodes_size=8000,
nodes_pos=bipart_node_pos).show(ax=ax)
SAS.pyplot(fig)
```



Performing ingredient-side projection

In this example, the number of common neighbors (recipe nodes) that are shared between pairs of ingredient nodes is of interest. To perform the projection, you must specify a subset of nodes; this subset indicates the nodes onto which the graph is to be projected. These nodes form the node set of the resulting projected graph.

Because the goal is to infer edges between pairs of ingredients, you specify the ingredient nodes for the projection as follows. As a result, two ingredient nodes are connected in the projected graph if they share one or more recipe nodes as common neighbors.

```
# Ingredient-side projection
ingredients = {i for l in adj.values() for i in l}
Gproj = G.projected_graph_common_neighbors(nodes=ingredients)
print(Gproj.to_pandas_edges())
```

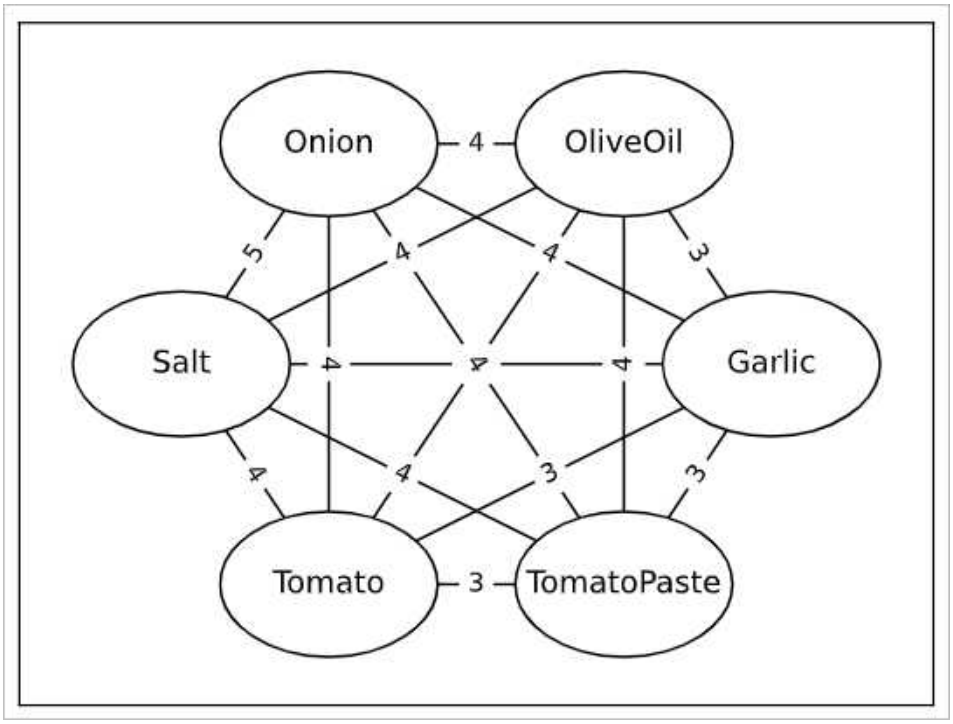
```
Graph object with 40 nodes and 58 edges
         from           to  common_neighbors
0     BayLeaf       Carrot                1.0
1     BayLeaf       Celery                1.0
2     BayLeaf       Garlic                1.0
3     BayLeaf  GreenPepper                1.0
4     BayLeaf   GroundBeef                1.0
..        ...          ...                ...
206  Cilantro    Tomatillo                1.0
207  Jalapeno         Lime                1.0
208  Jalapeno    Tomatillo                1.0
209      Lime    Tomatillo                1.0
210   Oregano      Parsley                1.0
[211 rows x 3 columns]
```

Visualizing the strong ingredient links

You use the following statements to visualize projected edges that have at least three common neighbors, because strongly linked ingredients are considered good candidates for combination in future recipes:

```
# Visualizing the strong ingredient links
```

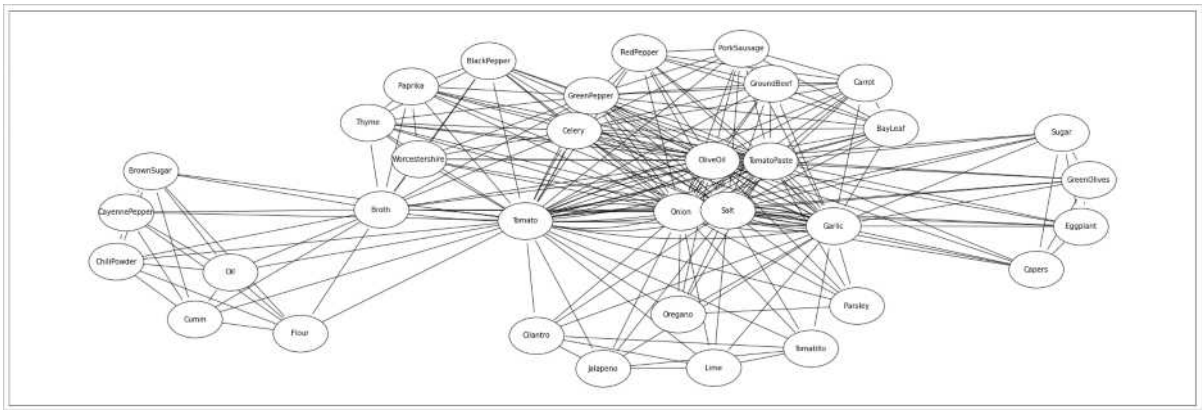
```
edges_to_retain = [(u,v) for (u,v,data) in Gproj.edges(data=True) if
data['common_neighbors']>=3]
Gproj_sub = sasnet.edge_subgraph(Gproj, edges_subset=edges_to_retain)
fig, ax = plt.subplots()
sasnet.PlotGraph(Gproj_sub, nodes_size=5000,
edges_label='common_neighbors').show(ax=ax)
SAS.pyplot(fig)
```



Creating and visualizing the projected multigraph

You create the ingredient-side projection again. This time you use the MULTIGRAPH=TRUE option as follows to produce a multigraph in which each edge represents a shared recipe (common neighbor) between two ingredients. You then visualize the multigraph to display all individual connections.

```
# Creating and visualizing the projected multigraph
fig, ax = plt.subplots(figsize=(36,12))
Gproj_multi = G.projected_graph(nodes=ingredients, multigraph=True)
sasnet.PlotGraph(Gproj_multi, nodes_size=6000).show(ax=ax)
SAS.pyplot(fig)
```



Visualizing the recipes that contain both garlic and olive oil

Then you visualize recipes that contain both garlic and olive oil. You do this as follows by creating a subgraph that includes the ('OliveOil','Garlic') edge. Then you simplify this edge by combining all shared neighbor labels into a single string, which is displayed as the edge label in the visualization.

```
# Visualizing the recipes that contain both Garlic and OliveOil
G_sub = sasnet.edge_subgraph(Gproj_multi,
edges_subset=[('OliveOil','Garlic')])
G_simple = sasnet.simplify(G_sub, how={'neighbor':lambda
x:"\n".join(x)})

fig, ax = plt.subplots(figsize=(5,4))
sasnet.PlotGraph(G_simple, nodes_size=3000,
edges_label='neighbor').show(ax=ax)
SAS.pyplot(fig)
```