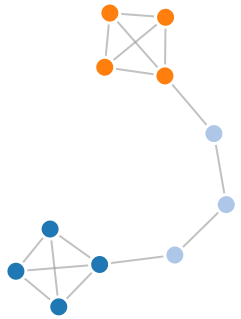
NetworkX

Network Analysis in Python

NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.



Software for complex networks

- Data structures for graphs, digraphs, and multigraphs
- Many standard graph algorithms
- Network structure and analysis measures
- Generators for classic graphs, random graphs, and synthetic networks
- Nodes can be "anything" (e.g., text, images, XML records)
- Edges can hold arbitrary data (e.g., weights, time-series)
- Open source 3-clause BSD license
- Well tested with over 90% code coverage
- Additional benefits from Python include fast prototyping, easy to teach, and multi-platform

Algorithms

A closer look at some of the algorithms and network analysis techniques provided by NetworkX.

Node assortativity coefficients and correlation measures

Directed Acyclic Graphs & Topological Sort

Dinitz's algorithm and its applications

Lowest Common Ancestor

Euler's Algorithm

Isomorphism - How to find if two graphs are similar?

Welcome to nx-guides!

Contents

This site provides educational materials officially developed and curated by the NetworkX community. The goal of the repository is to provide high-quality educational resources for learning about network analysis and graph theory with NetworkX. Examples include:

- Long-form narrative documentation, such as tutorials
- In-depth examinations of common graph and network algorithms and their implementations in NetworkX
- Demonstrations or domain-specific applications of NetworkX highlighting best-practices for network analysis.

About

The educational materials are in the form of markdown-based Jupyter notebooks, so everything is interactive! You can follow along yourself:

1. *on binder*, by clicking on the launch button at the top of this page, or the rocket icon in the upper-right corner of any

of the pages, or

2. *locally*, by cloning the repository (see the octocat icon above) and running `jupyter notebook`.

Contents

Graph Generators

A closer look at the functions provided by NetworkX to create interesting graphs.

Section Navigation

Reference

Release: 3.1
Date: Apr 04, 2023

Operators

Planarity

Planar Drawing

Graph Polynomials

Reciprocity

Regular

Rich Club

Shortest Paths

Similarity Measures

Simple Paths

Small-world

s metric

Sparsifiers

Structural holes

Summarization

Swap

Threshold Graphs

Tournament

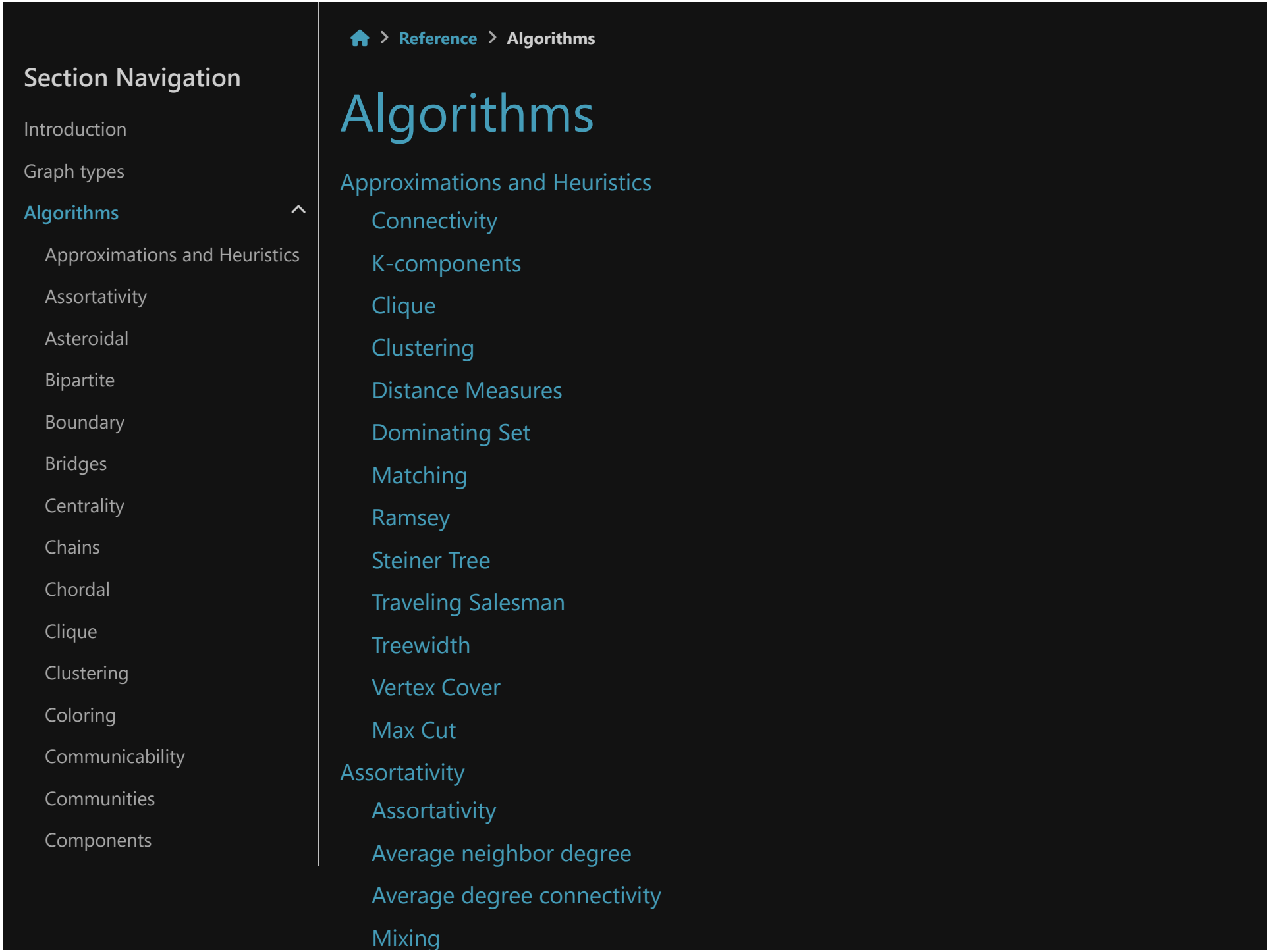
Traversal

Tree

Triads

Vitality

Specific List of Algorithms in Detail



Section Navigation

Introduction

Graph types

Algorithms

Approximations and Heuristics

Assortativity

Asteroidal

Bipartite

Boundary

Bridges

Centrality

Chains

Chordal

Clique

Clustering

Coloring

Communicability

Communities

Components

Reference > Algorithms

Algorithms

Approximations and Heuristics

Connectivity

K-components

Clique

Clustering

Distance Measures

Dominating Set

Matching

Ramsey

Steiner Tree

Traveling Salesman

Treewidth

Vertex Cover

Max Cut

Assortativity

Assortativity

Average neighbor degree

Average degree connectivity

Mixing

- local_bridges
- Centrality
  - Degree
  - Eigenvector
  - Closeness
  - Current Flow Closeness
  - (Shortest Path) Betweenness
  - Current Flow Betweenness
  - Communicability Betweenness
  - Group Centrality
  - Load
  - Subgraph
  - Harmonic Centrality
  - Dispersion
  - Reaching
  - Percolation
  - Second Order Centrality
  - Trophic
  - VoteRank
  - Laplacian
- Chains
  - chain_decomposition
- Chordal

- is_chordal
- chordal_graph_cliques
- chordal_graph_treewidth
- complete_to_chordal_graph
- find_induced_nodes

Clique

- enumerate_all_cliques
- find_cliques
- find_cliques_recursive
- make_max_clique_graph
- make_clique_bipartite
- graph_clique_number
- graph_number_of_cliques
- node_clique_number
- number_of_cliques
- cliques_containing_node
- max_weight_clique

Clustering

- triangles
- transitivity
- clustering
- average_clustering
- square_clustering

- generalized_degree
- Coloring
  - greedy_color
  - equitable_color
  - strategy_connected_sequential
  - strategy_connected_sequential_dfs
  - strategy_connected_sequential_bfs
  - strategy_independent_set
  - strategy_largest_first
  - strategy_random_sequential
  - strategy_saturation_largest_first
  - strategy_smallest_last
- Communicability
  - communicability
  - communicability_exp
- Communities
  - Bipartitions
  - K-Clique
  - Modularity-based communities
  - Tree partitioning
  - Label propagation
  - Louvain Community Detection

- k_core
- k_shell
- k_crust
- k_corona
- k_truss
- onion_layers

Covering
- min_edge_cover
- is_edge_cover

Cycles
- cycle_basis
- simple_cycles
- recursive_simple_cycles
- find_cycle
- minimum_cycle_basis
- chordless_cycles

Cuts
- boundary_expansion
- conductance
- cut_size
- edge_expansion
- mixing_expansion
- node_expansion

- node_expansion
- normalized_cut_size
- volume

D-Separation

- Blocking paths
- Illustration of D-separation with examples
- D-separation and its applications in probability
- Examples
- References
- d_separated

Directed Acyclic Graphs

- ancestors
- descendants
- topological_sort
- topological_generations
- all_topological_sorts
- lexicographical_topological_sort
- is_directed_acyclic_graph
- is_aperiodic
- transitive_closure
- transitive_closure_dag
- transitive_reduction
- antichains

- dag_longest_path
- dag_longest_path_length
- dag_to_branching

Distance Measures

- barycenter
- center
- diameter
- eccentricity
- periphery
- radius
- resistance_distance

Distance-Regular Graphs

- is_distance_regular
- is_strongly_regular
- intersection_array
- global_parameters

Dominance

- immediate_dominators
- dominance_frontiers

Dominating Sets

- dominating_set
- is_dominating_set

Efficiency

- efficiency
- local_efficiency
- global_efficiency

Eulerian

- is_eulerian
- eulerian_circuit
- eulerize
- is_semieulerian
- has_eulerian_path
- eulerian_path

Flows

- Maximum Flow
- Edmonds-Karp
- Shortest Augmenting Path
- Preflow-Push
- Dinitz
- Boykov-Kolmogorov
- Gomory-Hu Tree
- Utils
- Network Simplex
- Capacity Scaling Minimum Cost Flow

Graph Hashing

- weisfeiler_lehman_graph_hash

- weisfeiler_lehman_subgraph_hashes
- Graphical degree sequence
  - is_graphical
  - is_digraphical
  - is_multigraphical
  - is_pseudographical
  - is_valid_degree_sequence_havel_hakimi
  - is_valid_degree_sequence_erdos_gallai
- Hierarchy
  - flow_hierarchy
- Hybrid
  - kl_connected_subgraph
  - is_kl_connected
- Isolates
  - is_isolate
  - isolates
  - number_of_isolates
- Isomorphism
  - is_isomorphic
  - could_be_isomorphic
  - fast_could_be_isomorphic
  - faster_could_be_isomorphic

- VF2++
- Tree Isomorphism
- Advanced Interfaces

Link Analysis

- PageRank
- Hits

Link Prediction

- resource_allocation_index
- jaccard_coefficient
- adamic_adar_index
- preferential_attachment
- cn_soundarajan_hopcroft
- ra_index_soundarajan_hopcroft
- within_inter_cluster
- common_neighbor_centrality

Lowest Common Ancestor

- all_pairs_lowest_common_ancestor
- tree_all_pairs_lowest_common_ancestor
- lowest_common_ancestor

Matching

- is_matching
- is_maximal_matching
- is_perfect_matching

- maximal_matching
- max_weight_matching
- min_weight_matching

Minors

- References
- contracted_edge
- contracted_nodes
- identified_nodes
- equivalence_classes
- quotient_graph

Maximal independent set

- maximal_independent_set

non-randomness

- non_randomness

Moral

- moral_graph

Node Classification

- References
- harmonic_function
- local_and_global_consistency

Operators

- complement
- reverse

compose
union
disjoint_union
intersection
difference
symmetric_difference
full_join
compose_all
union_all
disjoint_union_all
intersection_all
cartesian_product
lexicographic_product
rooted_product
strong_product
tensor_product
power
corona_product

Planarity

check_planarity
is_planar
networkx.algorithms.planarity.PlanarEmbedding

Planar Drawing

Planar Drawing
combinatorial_embedding_to_pos
Graph Polynomials
tutte_polynomial
chromatic_polynomial
Reciprocity
reciprocity
overall_reciprocity
Regular
is_regular
is_k_regular
k_factor
Rich Club
rich_club_coefficient
Shortest Paths
shortest_path
all_shortest_paths
shortest_path_length
average_shortest_path_length
has_path
Advanced Interface
Dense Graphs
A* Algorithm
Similarity Measures

Similarity Measures

- graph_edit_distance
- optimal_edit_paths
- optimize_graph_edit_distance
- optimize_edit_paths
- simrank_similarity
- panther_similarity
- generate_random_paths

Simple Paths

- all_simple_paths
- all_simple_edge_paths
- is_simple_path
- shortest_simple_paths

Small-world

- random_reference
- lattice_reference
- sigma
- omega

s metric

- s_metric

Sparsifiers

- spanner

Structural holes

- constraint
- effective_size
- local_constraint

Summarization

- dedensify
- snap_aggregation

Swap

- double_edge_swap
- directed_edge_swap
- connected_double_edge_swap

Threshold Graphs

- find_threshold_graph
- is_threshold_graph

Tournament

- hamiltonian_path
- is_reachable
- is_strongly_connected
- is_tournament
- random_tournament
- score_sequence

Traversal

- Depth First Search
- Breadth First Search

- Beam search
- Depth First Search on Edges
- Breadth First Search on Edges

Tree

- Recognition
- Branchings and Spanning Arborescences
- Encoding and decoding
- Operations
- Spanning Trees
- Decomposition
- Exceptions

Triads

- triadic_census
- random_triad
- triads_by_type
- triad_type
- is_triad
- all_triads
- all_triplets

Vitality

- closeness_vitality

Voronoi cells

- voronoi_cells

Created using Sphinx 6.1.3.

Built with the PyData Sphinx Theme 0.13.3.

Traversal

Depth First Search

Basic algorithms for depth-first searching the nodes of a graph.

`dfs_edges`(G[, source, depth_limit])	Iterate over edges in a depth-first-search (DFS).
`dfs_tree`(G[, source, depth_limit])	Returns oriented tree constructed from a depth-first-search from source.
`dfs_predecessors`(G[, source, depth_limit])	Returns dictionary of predecessors in depth-first-search from source.
`dfs_successors`(G[, source, depth_limit])	Returns dictionary of successors in depth-first-search from source.
`dfs_preorder_nodes`(G[, source, depth_limit])	Generate nodes in a depth-first-search pre-ordering starting at source.
`dfs_postorder_nodes`(G[, source, depth_limit])	Generate nodes in a depth-first-search post-ordering starting at source.
`dfs_labeled_edges`(G[, source, depth_limit])	Iterate over edges in a depth-first-search (DFS) labeled by type.

Breadth First Search

Basic algorithms for breadth-first searching the nodes of a graph.

bfs_predecessors

`bfs_predecessors(G, source, depth_limit=None, sort_neighbors=None)` [source]

Returns an iterator of predecessors in breadth-first-search from source.

Parameters:

G : *NetworkX graph*

source : *node*

Specify starting node for breadth-first search

depth_limit : *int, optional(default=len(G))*

Specify the maximum search depth

sort_neighbors : *function*

A function that takes the list of neighbors of given node as input, and returns an *iterator* over these neighbors but with custom ordering.

Returns:

pred: iterator

(node, predecessor) iterator where `predecessor` is the predecessor of `node` in a breadth first search starting from `source`.

See also

`bfs_tree`

bfs_edges

edge_bfs

Notes

Based on http://www.ics.uci.edu/~eppstein/PADS/BFS.py by D. Eppstein, July 2004. The modifications to allow depth limits based on the Wikipedia article "Depth-limited-search".

Examples

```
>>> G = nx.path_graph(3)
>>> print(dict(nx.bfs_predecessors(G, 0)))
{1: 0, 2: 1}
>>> H = nx.Graph()
>>> H.add_edges_from([(0, 1), (0, 2), (1, 3), (1, 4), (2, 5), (2, 6)])
>>> print(dict(nx.bfs_predecessors(H, 0)))
{1: 0, 2: 0, 3: 1, 4: 1, 5: 2, 6: 2}
>>> M = nx.Graph()
>>> nx.add_path(M, [0, 1, 2, 3, 4, 5, 6])
>>> nx.add_path(M, [2, 7, 8, 9, 10])
>>> print(sorted(nx.bfs_predecessors(M, source=1, depth_limit=3)))
[(0, 1), (2, 1), (3, 2), (4, 3), (7, 2), (8, 7)]
>>> N = nx.DiGraph()
>>> nx.add_path(N, [0, 1, 2, 3, 4, 7])
>>> nx.add_path(N, [3, 5, 6, 7])
>>> print(sorted(nx.bfs_predecessors(N, source=2)))
[(3, 2), (4, 3), (5, 3), (6, 5), (7, 4)]
```