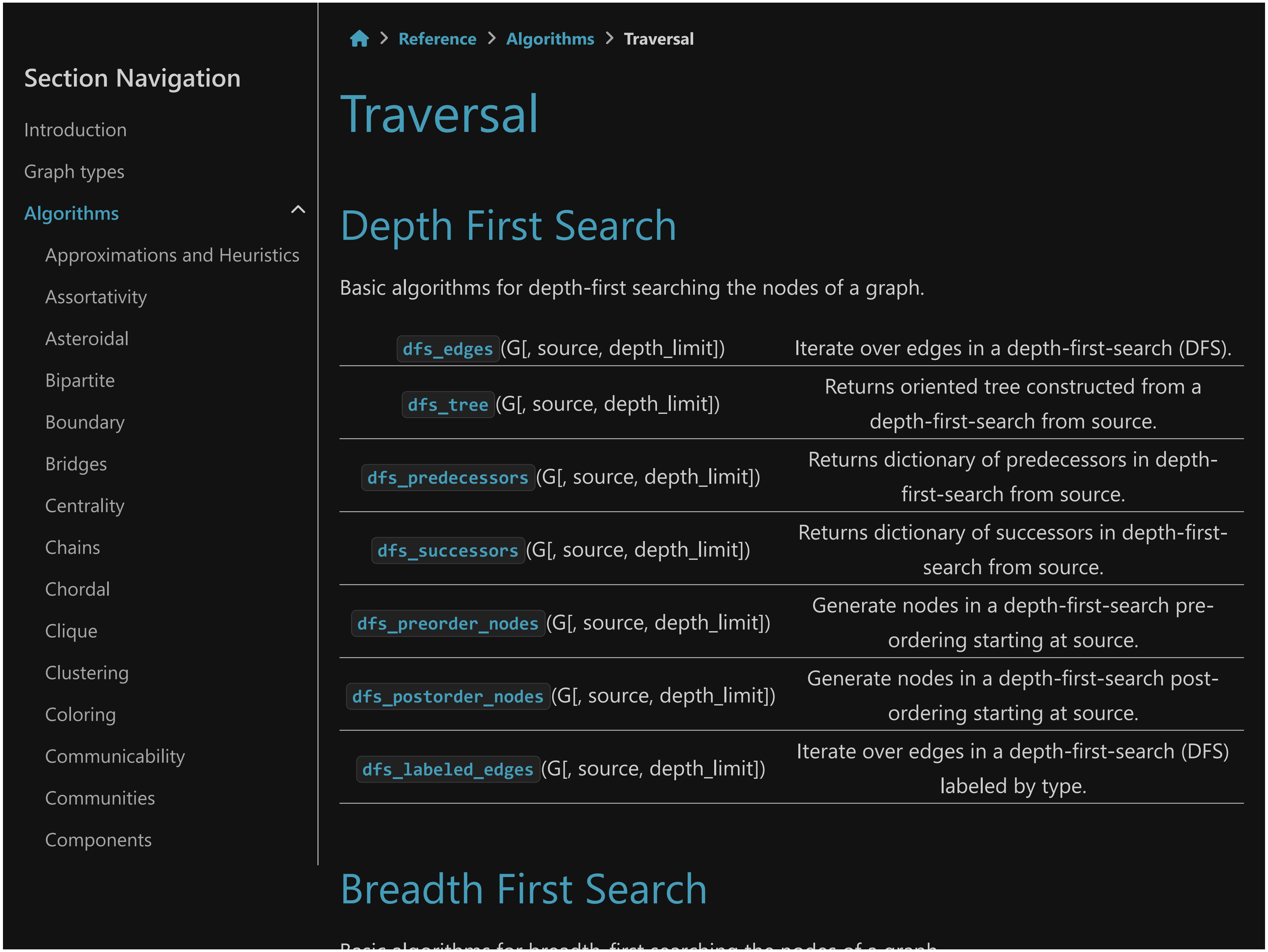


Traversal

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.

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

`bfs_edges`(G, source[, reverse, depth_limit, ...])	Iterate over edges in a breadth-first-search starting at source.
`bfs_layers`(G, sources)	Returns an iterator of all the layers in breadth-first search traversal.
`bfs_tree`(G, source[, reverse, depth_limit, ...])	Returns an oriented tree constructed from of a breadth-first-search starting at source.
`bfs_predecessors`(G, source[, depth_limit, ...])	Returns an iterator of predecessors in breadth-first-search from source.
`bfs_successors`(G, source[, depth_limit, ...])	Returns an iterator of successors in breadth-first-search from source.
`descendants_at_distance`(G, source, distance)	Returns all nodes at a fixed `distance` from `source` in `G`.

Beam search

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

`bfs_beam_edges`(G, source, value[, width])	Iterates over edges in a beam search.

Depth First Search on Edges

Algorithms for a depth-first traversal of edges in a graph.

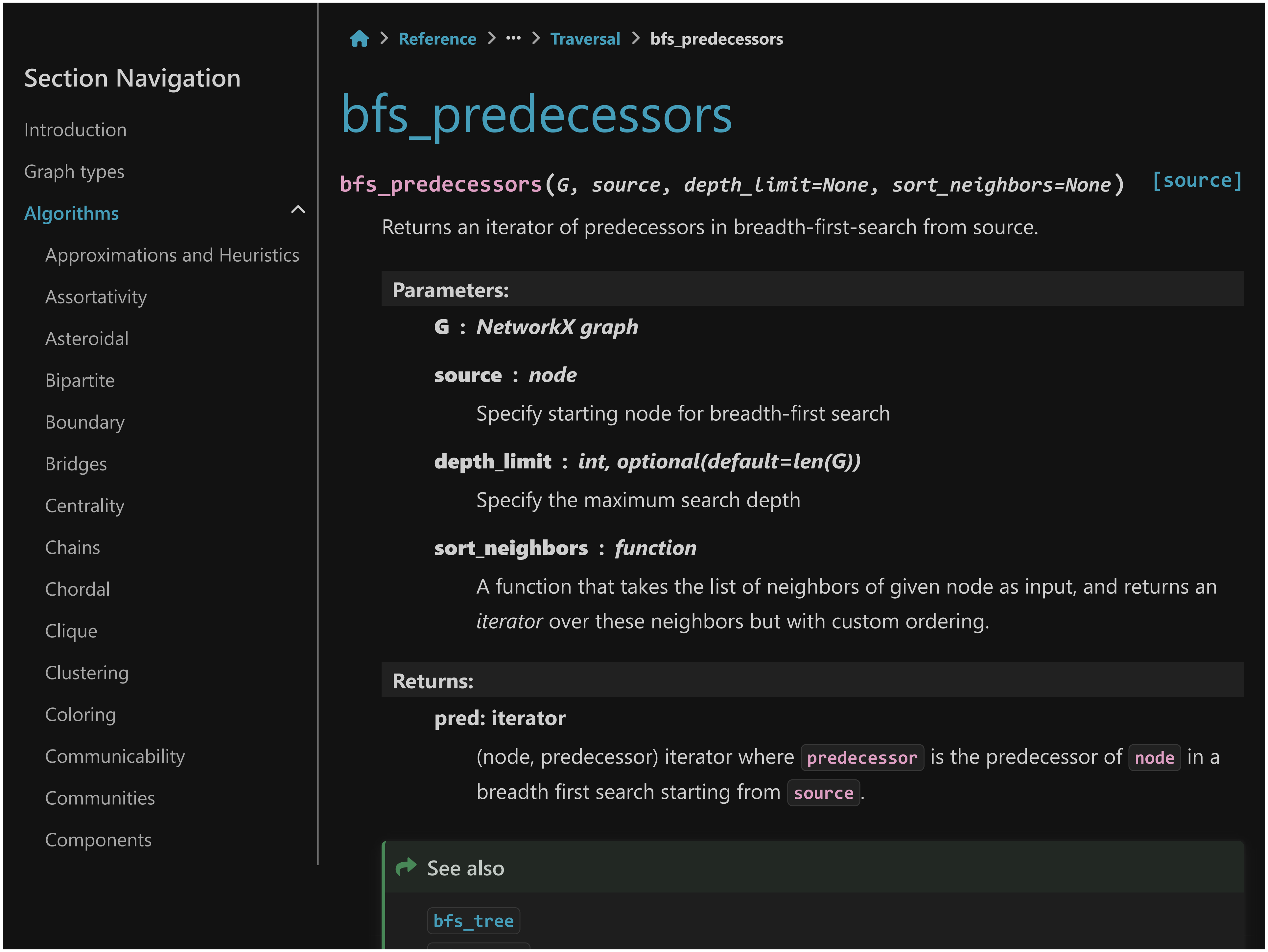
`edge_dfs`(G[, source, orientation])	A directed, depth-first-search of edges in `G`, beginning at

`source`.

Breadth First Search on Edges

Algorithms for a breadth-first traversal of edges in a graph.

`edge_bfs`(G[, source, orientation])	A directed, breadth-first-search of edges in `G`, beginning at `source`.



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)]
```