Package: crann (via r-universe)

May 31, 2026

Title Spanning Tree Methods for Graphs

Version 0.0.1

Description Implements spanning tree algorithms for undirected graphs represented as 'adj' adjacency lists. Provides enumeration of all spanning trees via Winter's (1986) contraction-based algorithm <doi:10.1007/BF01939361>, counting via Kirchhoff's matrix tree theorem, minimum spanning trees via Kruskal's algorithm, uniform random sampling via Wilson's (1996) loop-erased random walk <doi:10.1145/237814.237880>, and structural utilities including fundamental cycles and cuts.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Imports adj, cli

Config/testthat/edition 3

Config/Needs/website christopherkenny/ctktemplate

Suggests spelling, testthat (>= 3.0.0)

Language en-US

URL http://christophertkenny.com/crann/

Repository https://christopherkenny.r-universe.dev

Date/Publication 2026-05-01 16:14:53 UTC

RemoteUrl https://github.com/christopherkenny/crann

RemoteRef HEAD

RemoteSha d1bb3933396a168765b6a630747467e517a321e1

Contents

`count_spanning_trees` *Count spanning trees*

Description

Counts the number of spanning trees of a connected undirected graph using Kirchhoff's matrix tree theorem: the count equals the determinant of any (n-1) x (n-1) cofactor of the graph Laplacian.

Usage

```
count_spanning_trees(graph)
```

Arguments

`graph`	An `adj` object representing a connected undirected graph.

Value

A `numeric` scalar. (Integer-valued but returned as `numeric` since counts can exceed `.Machine$integer.max` for dense graphs.)

See Also

`enumerate_spanning_trees()` to list all spanning trees.

Examples

```
# Triangle: 3 spanning trees
g <- adj::adj(list(c(2L, 3L), c(1L, 3L), c(1L, 2L)))
count_spanning_trees(g)
```

enumerate_spanning_trees

Enumerate all spanning trees

Description

Enumerates all spanning trees of a connected undirected graph using Winter's (1986) contraction-based algorithm. The algorithm has worst-case time complexity O(n + m + nt) and space complexity O(n^2), where n is the number of vertices, m the number of edges, and t the number of spanning trees.

Usage

```
enumerate_spanning_trees(graph)
```

Arguments

graph	An adj object representing a connected undirected graph without self-loops or duplicate edges.

Value

A list of adj objects, one per spanning tree.

References

Winter, P. (1986). An algorithm for the enumeration of spanning trees. *BIT Numerical Mathematics*, 26(1), 44–62. doi:10.1007/BF01939361

Examples

```
# Triangle: 3 spanning trees
g <- adj::adj(list(c(2L, 3L), c(1L, 3L), c(1L, 2L)))
trees <- enumerate_spanning_trees(g)
length(trees)
```

enumerate_spanning_trees_edges

Enumerate spanning trees as an edge matrix

Description

Returns a compact integer matrix encoding all spanning trees of a connected undirected graph. The matrix has n-1 rows and 2t columns, where n is the number of vertices and t is the number of spanning trees. For spanning tree k (1-indexed), columns 2k-1 and 2k hold the u and v endpoints (1-indexed vertex IDs) of each of the n-1 edges.

Usage

```
enumerate_spanning_trees_edges(graph)
```

Arguments

graph	An adj object representing a connected undirected graph without self-loops or duplicate edges.

Details

This function is substantially faster than enumerate_spanning_trees() for graphs with many spanning trees because it avoids per-tree R memory allocation: the entire output is a single integer matrix allocation.

Value

An integer matrix with n-1 rows and 2t columns.

References

Winter, P. (1986). An algorithm for the enumeration of spanning trees. *BIT Numerical Mathematics*, 26(1), 44–62. doi:10.1007/BF01939361

Examples

```
# Triangle: 3 spanning trees, 2 edges each -> 2 x 6 matrix
g <- adj::adj(list(c(2L, 3L), c(1L, 3L), c(1L, 2L)))
enumerate_spanning_trees_edges(g)
```

---

fundamental_cuts *Fundamental cuts of a spanning tree*

Description

Returns the fundamental cuts of a spanning tree with respect to the original graph. There is one fundamental cut per tree edge (n - 1 total), consisting of all edges in the graph that cross the bipartition induced by removing that tree edge.

Usage

```
fundamental_cuts(graph, tree)
```

Arguments

graph	An adj object representing a connected undirected graph.
tree	An adj object representing a spanning tree of graph.

Value

A list of `adj` objects, one per tree edge. Each `adj` represents the cut as a subgraph of `graph`.

See Also

`fundamental_cycles()`

Examples

```
g <- adj::adj(list(c(2L, 3L), c(1L, 3L), c(1L, 2L)))
t <- minimum_spanning_tree(g)
fundamental_cuts(g, t)
```

---

fundamental_cycles *Fundamental cycles of a spanning tree*

Description

Returns the fundamental cycles of a spanning tree with respect to the original graph. There is one fundamental cycle per non-tree edge (m - n + 1 total), formed by adding that edge to the unique path between its endpoints in the spanning tree.

Usage

```
fundamental_cycles(graph, tree)
```

Arguments

`graph`	An `adj` object representing a connected undirected graph.
`tree`	An `adj` object representing a spanning tree of `graph`.

Value

A list of `adj` objects, one per non-tree edge. Each `adj` represents the cycle as a subgraph of `graph`.

See Also

`fundamental_cuts()`

Examples

```
g <- adj::adj(list(c(2L, 3L), c(1L, 3L), c(1L, 2L)))
t <- minimum_spanning_tree(g)
fundamental_cycles(g, t)
```

is_spanning_tree *Test if a graph is a spanning tree*

Description

Returns TRUE if tree is a spanning tree: a connected acyclic graph with exactly n-1 edges. To additionally check that tree uses only edges from a reference graph, see is_spanning_tree_of().

Usage

```
is_spanning_tree(tree)
```

Arguments

tree	An adj object to test.

Value

A logical scalar.

See Also

is_spanning_tree_of(), minimum_spanning_tree(), enumerate_spanning_trees()

Examples

```
g <- adj::adj(list(c(2L, 3L), c(1L, 3L), c(1L, 2L)))
t <- minimum_spanning_tree(g)
is_spanning_tree(t)
```

is_spanning_tree_of *Test if a graph is a spanning tree of another graph*

Description

Returns TRUE if tree is a spanning tree of graph: a connected acyclic subgraph that spans all vertices and uses only edges present in graph.

Usage

```
is_spanning_tree_of(tree, graph)
```

Arguments

tree	An adj object to test.
graph	An adj object representing a connected undirected graph.

Value

A logical scalar.

See Also

`is_spanning_tree()`, `minimum_spanning_tree()`, `enumerate_spanning_trees()`

Examples

```
g <- adj::adj(list(c(2L, 3L), c(1L, 3L), c(1L, 2L)))
t <- minimum_spanning_tree(g)
is_spanning_tree_of(t, g)
```

---

minimum_spanning_tree *Minimum spanning tree*

Description

Finds a minimum spanning tree of a connected undirected graph using Kruskal's algorithm. When `weights` is `NULL` (the default), all edges are treated as having equal weight and any spanning tree is returned.

Usage

```
minimum_spanning_tree(graph, weights = NULL)
```

Arguments

`graph`	An `adj` object representing a connected undirected graph.
`weights`	A numeric vector of edge weights, one per edge. Edges are ordered by iterating vertices `u = 1, ..., n` and for each `u` iterating over `graph[[u]]` retaining only neighbors `v > u`. Pass `NULL` for uniform weights.

Value

An `adj` object representing the minimum spanning tree.

See Also

`is_spanning_tree()` to validate a spanning tree.

Examples

```
g <- adj::adj(list(c(2L, 3L), c(1L, 3L), c(1L, 2L)))
minimum_spanning_tree(g)
```

`sample_spanning_tree` *Sample a uniform random spanning tree*

Description

Samples a spanning tree uniformly at random from the set of all spanning trees using Wilson's (1996) loop-erased random walk algorithm.

Usage

```
sample_spanning_tree(graph)
```

Arguments

`graph`	An `adj` object representing a connected undirected graph.

Value

An `adj` object representing a spanning tree of `graph`.

References

Wilson, D.B. (1996). Generating random spanning trees more quickly than the cover time. *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, 296–303. doi:10.1145/237814.237880

See Also

`enumerate_spanning_trees()`, `minimum_spanning_tree()`

Examples

```
set.seed(1)
g <- adj::adj(list(c(2L, 3L), c(1L, 3L), c(1L, 2L)))
sample_spanning_tree(g)
```

Index