Package 'rsppfp'

July 23, 2025

Title R's Shortest Path Problem with Forbidden Subpaths

Version 1.0.4

Maintainer Melina Vidoni <melinavidoni@santafe-conicet.gov.ar>

Description

An implementation of functionalities to transform directed graphs that are bound to a set of known forbidden paths. There are several transformations, following the rules provided by Villeneuve
and De-
saulniers (2005) <doi:10.1016/j.ejor.2004.01.032>, and Hsu et al. (2009) <doi:10.1007/978-3-642-03095-6_60>.
The resulting graph is generated in a data-frame format. See rsppfp website for more information, documentation an examples.

Depends R (>= 3.4.0)

Imports dplyr, foreach, doParallel, igraph, tidyr, stringr

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Suggests knitr, rmarkdown, testthat, covr, ggplot2

VignetteBuilder knitr

URL https://github.com/melvidoni/rsppfp

BugReports https://github.com/melvidoni/rsppfp/issues

NeedsCompilation no

Author Melina Vidoni [aut, cre] (ORCID:
<https://orcid.org/0000-0002-4099-1430>),
Aldo Vecchietti [aut]

Repository CRAN

Date/Publication 2019-02-19 15:10:03 UTC

Contents

`direct_graph` *Undirected Graph Translator*

Description

The SPPFP transformation functions only work with digraphs -i.e. directed graphs. Because in a digraph arcs can only be traveled in one direction, from the origin node to the destination arc, if undirected graphs are used as-is, the resultng G* will not be accurate. Therefore, this functions translates an undirected graph to a digraph by duplicating each arc and swapping the duplicate's `from` and `to` nodes.

Usage

```
direct_graph(graph, cores = 1L)
```

Arguments

`graph`	An undirected graph written as a data frame, in which each rows represent an arc. The columns must be named `from` and `to`, and can be of any data type. Each row can have additional attributes, and no cells can be `NULL` or `NA`.
`cores`	This algorithm can be run using R's parallel processing functions. This variable represents the number of processing cores you want to assign for the transformation. The default value is one single core. It is suggested to not assign all of your available cores to the function.

Value

A new graph, with the same columns and data types of the original graph. This new graph is twice as big as the original, as new arcs are added to represent that each arc can be traveled in both directions.

See Also

Other Parsers: `get_all_nodes`, `parse_vpath`

Examples

```
# Obtain the graph from any way
graph <- data.frame(from = c("s", "s", "s", "u", "u", "w", "w", "x", "x", "v", "v", "y", "y"),
                    to = c("u", "w", "x", "w", "v", "v", "y", "w", "y", "y", "t", "t", "u"),
                       cost = c(1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L),
                       stringsAsFactors = FALSE)
graph

# Translate it
digraph <- direct_graph(graph)
digraph
```

---

get_all_nodes *Parser for G* nodes.*

Description

A original node N_i can appear on a transformed G* as different nodes. This is the result of the creation of nodes in the transformation processes. Therefore, it is possible that the original node N does not exists on G*, or that multiple N_i* exist. Hence, as all new nodes are generated using a specific structure for the name -compiling all previous nodes names, split by pipe-, this function allows searching for all the N_i* nodes that are equivalente to N_i. This can be used to find shortest paths to all of them.

Usage

```
get_all_nodes(g, originalNode)
```

Arguments

g	A graph in data frame format, translated using one of the available functions.
originalNode	The name of the original node from G, that needs to be searched within G*. It is preferable to use a character format, but this can also be of any simple type. No lists or vectors are allowed.

Value

A new vector of character type, whose elements are all the N_i* equivalent to the original N node. This also includes the original node.

See Also

Other Parsers: direct_graph, parse_vpath

Examples

```
# Given a specific gStar graph:
gStar <- data.frame(from = c("u|v", "s|u|v", "s|u", "s", "s", "u", "w", "w", "x", "x", "v",
                             "v", "y", "y", "s", "s|u", "u", "u|v"),
                    to = c("t", "u|v|y", "w", "w", "x", "w", "v", "y", "w", "y", "y", "t",
                           "t", "u", "s|u", "s|u|v", "u|v", "u|v|y"),
                     weight = c(12L, 3L, 5L, 9L, 7L, 5L, 11L, 10L, 1L, 2L, 3L, 12L, 13L,
                                0L, 8L, 4L, 4L, 3L),
                     stringsAsFactors = FALSE)
gStar

# Obtain all the nodes equivalent to N_i = "v"
get_all_nodes(gStar, "v")
```

`get_shortest_path` *igraph Shortest Path*

Description

A original node N_i can appear on a transformed gStar as different N_i* equivalent nodes. Therefore, this becomes a limitation when searching for a shortest path inside gStar. As a result: all N_i* need to be considered as possible destination nodes when looking for the shortest path. This function is a wrapper for this behavior, providing a straightforward implementation using igraph capabilities. However, it aims to provide guidance on how to build a similar algorithm for different path-finding algorithms.

It is important to mention that new nodes are only considered as destination nodes, and they are not search for origin nodes. This is because N* nodes can only be reached after traveling through gStar nodes. For example, a node `"f|e|r"` is actually indicating that `"r"` has been reached after traveling through the nodes `"f"` and `"e"`.

Usage

```
get_shortest_path(g, origin, dest, weightColName = NULL)
```

Arguments

`g`	A gStar digraph in data frame format, translated using one of the available functions. The weight or cost attribute of each arc of the graph must be stored in a specific column named `weight`.
`origin`	The name of the starting node from G for the path. It must be written as it appears in G, and it is preferable to use a character format, but this can also be of any simple type. No lists or vectors are allowed.
`dest`	The name of the destination node from G for the path. It must be written as it appears in G, and it is preferable to use a character format, but this can also be of any simple type. No lists or vectors are allowed.

`weightColName` The name of the weight column used in the dataframe. If the column does not exist, a name _must_ be provided so that a new weight column is generated.

Value

The shortest path from `origin` node to `dest` node, calculated in G*, to include the forbidden paths. It uses igraph's functionalities.

Examples

```
# Given a specific gStar graph:
gStar <- data.frame(from = c("u|v", "s|u|v", "s|u", "s", "s", "u", "w", "w", "x", "x",
                             "v", "v", "y", "y", "s", "s|u", "u", "u|v"),
                    to = c("t", "u|v|y", "w", "w", "x", "w", "v", "y", "w", "y", "y", "t",
                           "t", "u", "s|u", "s|u|v", "u|v", "u|v|y"),
                   weight = c(12L, 3L, 5L, 9L, 7L, 5L, 11L, 10L, 1L, 2L, 3L, 12L, 13L, 0L,
                              8L, 4L, 4L, 3L),
                    stringsAsFactors = FALSE)
gStar

# Obtain the shortest path
get_shortest_path(gStar, "s", "v", "weight")
```

`modify_graph_hsu` *Hsu et al. (2009) Algorithm*

Description

It is an implementation of Hsu et al. algorithm to transform a digraph and a known set of forbidden paths, into a new graph that does not allow any forbidden path as part of its solutions.

Usage

```
modify_graph_hsu(g, f, cores = 1L)
```

Arguments

`g` The digraph to be transformed, written as a data frame where each row represents a directed arc. The columns must be named `from` and `to`, and can be of any data type. On each row no cells can be `NULL` or `NA`.

`f` The set of forbidden paths, written as a data frame. Each row represents a path as a sequence of nodes. Each row may be of different size, filling the empty cells with `NA`. All nodes involved must be part of `g`, and no forbidden path can be of size 2. This is because the latter is thought as an arc that should not exist in the first place.

`cores` This algorithm can be run using R's parallel processing functions. This variable represents the number of processing cores you want to assign for the transformation. The default value is one single core. It is suggested to not assign all of your available cores to the function.

Details

This version of the algorithm produce smaller graphs, with less new nodes and arcs.

Value

A new graph, generated following Hsu's backward construction, in which no path includes one of the forbidden subpaths. The graph is returned in a data frame format, where each row represents a directed arc, with or without additional attributes (if corresponds). However, regardless of the data type of the original graph, nodes on the new graph are of type character. The new nodes names are generated by incrementally concatenating the nodes on a forbidden path, but split by a pipe character (|).

See Also

https://doi.org/10.1007/978-3-642-03095-6_60

Other Graph Transformation: modify_graph_vd

Examples

```
# Obtain a graph and its forbidden subpaths
graph <- data.frame(from = c("c", "c", "u", "u", "t", "a", "a", "r", "e", "e", "e",
                             "p", "i", "i", "n", "o"),
                    to = c("u", "p", "e", "t", "a", "r", "i", "u", "r", "i", "p",
                           "n", "n", "o", "o", "m"),
                    stringsAsFactors = FALSE)
fpaths <- data.frame(X1 = c("u", "p", "a", "a"), X2 = c("t", "n", "i", "r"),
                     X3 = c("a", "o", "n", "u"), X4 = c("r", "m", "o", NA),
                     X5 = c("u", NA, NA, NA), stringsAsFactors = FALSE)

# Show the input
graph
fpaths

# Call the function and store the result
gStar <- modify_graph_hsu(graph, fpaths)
gStar
```

`modify_graph_vd` *Villeneuve and Desaulniers (2005) Algorithm*

Description

It is an implementation of Villeneuve and Desaulniers' algorithm to transform a digraph and a known set of forbidden paths, into a new graph that does not allow any forbidden path as part of its solutions. This algorithm should only be used when there is certainty that no forbidden path is a sub-path (or contains a sub-path) of another forbidden path.

Usage

```
modify_graph_vd(g, f, cores = 1L)
```

Arguments

`g`	The digraph to be transformed, written as a data frame where each row represents a directed arc. The first two columns must be named `from` and `to`, and can be of any data type. No cells can be `NULL` or `NA`.
`f`	The set of forbidden paths, written as a data frame. Each row represents a path as a sequence of nodes. Each row may be of different size, filling the empty cells with `NA`. All nodes involved must be part of `g`, and no forbidden path can be of size 2. This is because the latter is thought as an arc that should not exist in the first place. Also, no forbidden path can be sub-path (or contain a sub-path) of another forbidden path. The columns names are not relevant.
`cores`	This algorithm can be run using R's parallel processing functions. This variable represents the number of processing cores you want to assign for the transformation. The default value is one single core. It is suggested to not assign all of your available cores to the function.

Value

A new graph, generated following Villeneuve's prodcedure, in which no path includes one of the forbidden subpaths. The graph is returned in a data frame format, where each row represents a directed arc. However, regardless of the data type of the original graph, nodes on the new graph are of type character. The new nodes names are generated by incrementally concatenating the nodes on a forbidden path, but split by a pipe character (`|`). The new graph includes all of the additional attributes that the original graph had.

See Also

https://doi.org/10.1016/j.ejor.2004.01.032

Other Graph Transformation: `modify_graph_hsu`

Examples

```
# Obtain a graph and its forbidden subpaths
graph <- data.frame(from = c("s", "s", "s", "u", "u", "w", "w", "x", "x", "v", "v",
                             "y", "y"),
                    to = c("u", "w", "x", "w", "v", "v", "y", "w", "y", "y", "t", "t", "u"),
                       cost = c(1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L),
                       stringsAsFactors = FALSE)

fpaths <- data.frame(V1 = c("s", "u"), V2 = c("u", "v"), V3 = c("v", "y"), V4 = c("t", "u"),
                     stringsAsFactors = FALSE)

# Show the values
graph
fpaths

# Call the function and store the result
modify_graph_vd(graph, fpaths)
```

---

parse_vpath *Parser for G* nodes paths.*

Description

Translates a sequence of nodes from a G* graph, generated with any of the available transformations, to a sequence of nodes in terms of the original G.

Usage

```
parse_vpath(vpath)
```

Arguments

vpath	A vector of character type, representing a path as a sequence of nodes. The nodes are supposed to belong to an original graph G, but be written in terms of G*.

Value

A new vector of character type, representing the same path as `vpath` but with the nodes names translated to the original graph G's names.

See Also

Other Parsers: `direct_graph`, `get_all_nodes`

Examples

```
# Obtain the vpath from any way, an algorithm or random walk.
# Call the parsing function
translated_vpath <- parse_vpath( c("s|u", "s|u|v", "u|v|y", "t") )

# Print the result
translated_vpath
```

---

rsppfp *Package: rsppfp*

Description

Transformation algorithms to translate the SPPFP (Shortest Path Problem with Forbidden Paths) to a traditional shortest-path problem that includes the forbidden paths.

Details

The SPPFP is a variant of the traditional shortest path problem, in which no solution can include any path listed on a known set of forbidden paths. The current approach to solve this is to translate the existing graph, and its set of forbidden paths, to a graph in which no path will include any forbidden sequence. This package provides straightforward parallel processing capabilities, as well as translation functions to use the algorithms on undirected graphs. It is highly compatible with other network research packages, as it only uses native R data types.

Index