Package 'cppRouting'

June 21, 2019

Type Package

Title Fast Implementation of Dijkstra Algorithm

Version 1.1

Date 2019-06-15

Author Vincent Larmet

Maintainer Vincent Larmet <larmet.vincent@gmail.com>

Description Calculation of distances, shortest paths and isochrones on weighted graphs using several variants of Dijkstra algorithm.
Proposed algorithms are unidirectional Dijkstra (Dijkstra, E. W. (1959) <doi:10.1007/BF01386390>),
bidirectional Dijkstra (Goldberg, Andrew & Fonseca F. Werneck, Renato (2005) <https://pdfs.semanticscholar.org/0761/18dfbe1d5a220f6ac59b4de4ad07b50283ac.pdf>),
A* search (P. E. Hart, N. J. Nilsson et B. Raphael (1968) <doi:10.1109/TSSC.1968.300136>),
new bidirectional A* (Pijls & Post (2009) <http://repub.eur.nl/pub/16100/ei2009-10.pdf>).

License GPL (>= 2)

Imports Rcpp (>= 1.0.1), parallel

LinkingTo Rcpp

SystemRequirements GNU make, C++11

RoxygenNote 6.1.1

URL https://github.com/vlarmet/cppRouting

Suggests knitr, rmarkdown, igraph

VignetteBuilder knitr

NeedsCompilation yes

Repository CRAN

Date/Publication 2019-06-21 16:30:12 UTC

R topics documented:

`cppRouting-package` *Fast Implementation of Dijkstra Algorithm in R*

Description

Calculation of distances, shortest paths and isochrones on non-negative weighted graphs using several variants of Dijkstra algorithm.

Functions

- distance matrix (between all combinations origin-destination nodes),
- distances between origin and destination by pair (one-to-one),
- shortest paths between origin and destination by pair (one-to-one),
- shortest paths between all origin nodes and all destination nodes,
- Isochrones/isodistances with one or multiple breaks.

Algorithms

Algorithms can be chosen for one-to-one calculations like get_distance_pair() and get_path_pair() :

- uni-directional Dijkstra algorithm,
- bi-directional Dijkstra algorithm,
- uni-directional A* algorithm (nodes coordinates are needed),
- New bi-directional A* algorithm (nodes coordinates are needed).

References

Dijkstra, E. W. (1959), A note on two problems in connexion with graphs.

P. E. Hart, N. J. Nilsson et B. Raphael (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths.

Goldberg, Andrew & Fonseca F. Werneck, Renato (2005). Computing Point-to-Point Shortest Paths from External Memory.

Pijls & Post (2009). Yet another bidirectional algorithm for shortest paths.

get_distance_matrix *Compute all shortest distance between origin and destination nodes.*

Description

Compute all shortest distance between origin and destination nodes.

Usage

```
get_distance_matrix(Graph, from, to, allcores = FALSE)
```

Arguments

Graph	An object generated by cppRouting::makegraph() function.
from	A vector of one or more vertices from which distances are calculated (origin).
to	A vector of one or more vertices (destination).
allcores	Logical. If TRUE, all cores are used.

Value

Matrix of shortest distances.

Note

get_distance_matrix() recursively perform Dijkstra algorithm for each 'from' nodes.

Examples

```
#Data describing edges of the graph
edges<-data.frame(from_vertex=c(0,0,1,1,2,2,3,4,4),
                  to_vertex=c(1,3,2,4,4,5,1,3,5),
                  cost=c(9,2,11,3,5,12,4,1,6))
#Get all nodes
nodes<-unique(c(edges$from_vertex,edges$to_vertex))

#Construct directed and undirected graph
directed_graph<-makegraph(edges,directed=TRUE)
non_directed<-makegraph(edges,directed=FALSE)

#Get matrix of distance between all nodes in the two graphs
dir_dist<-get_distance_matrix(Graph=directed_graph, from=nodes, to=nodes, allcores=FALSE)
non_dir_dist<-get_distance_matrix(Graph=non_directed, from=nodes, to=nodes, allcores=FALSE)
print(dir_dist)
print(non_dir_dist)
```

get_distance_pair *Compute shortest distance between origin and destination nodes.*

Description

Compute shortest distance between origin and destination nodes.

Usage

```
get_distance_pair(Graph, from, to, algorithm = "Dijkstra",
  constant = 1, allcores = FALSE)
```

Arguments

Graph	An object generated by cppRouting::makegraph() function.
from	A vector of one or more vertices from which distances are calculated (origin).
to	A vector of one or more vertices (destination).
algorithm	character. "Dijkstra" for uni-directional Dijkstra, "bi" for bi-directional Dijkstra, "A*" or "NBA" for New bi-directional A star .Default to "Dijkstra"
constant	numeric. Constant to maintain the heuristic function admissible in A* algorithm. Default to 1, when cost is expressed in the same unit than coordinates. See details
allcores	Logical. If TRUE, all cores are used.

Details

To perform A*, projected coordinates should be provided in the Graph object. In A* algorithms, euclidean distance is used as heuristic function. To understand how A star algorithm work, see https://en.wikipedia.org/wiki/A*_search_algorithm . To understand the importance of constant parameter, see the package description : https://github.com/vlarmet/cppRouting

Value

Vector of shortest distances.

Note

'from' and 'to' must be the same length.

Examples

```
#Data describing edges of the graph
edges<-data.frame(from_vertex=c(0,0,1,1,2,2,3,4,4),
                  to_vertex=c(1,3,2,4,4,5,1,3,5),
                  cost=c(9,2,11,3,5,12,4,1,6))

#Get all nodes
```

```
nodes<-unique(c(edges$from_vertex,edges$to_vertex))

#Construct directed and undirected graph
directed_graph<-makegraph(edges,directed=TRUE)
non_directed<-makegraph(edges,directed=FALSE)

#Sampling origin and destination nodes
origin<-sample(nodes,10,replace=TRUE)
destination<-sample(nodes,10,replace=TRUE)

#Get distance between origin and destination in the two graphs
dir_dist<-get_distance_pair(Graph=directed_graph, from=origin, to=destination, allcores=FALSE)
non_dir_dist<-get_distance_pair(Graph=non_directed, from=origin, to=destination, allcores=FALSE)
print(dir_dist)
print(non_dir_dist)
```

`get_isochrone` *Compute isochrones/isodistances from nodes.*

Description

Compute isochrones/isodistances from nodes.

Usage

```
get_isochrone(Graph, from, lim, setdif = FALSE)
```

Arguments

`Graph`	An object generated by cppRouting::makegraph() function.
`from`	numeric or character. A vector of one or more vertices from which isochrones/isodistances are calculated.
`lim`	numeric. A vector of one or multiple breaks.
`setdif`	logical. If TRUE and length(lim)>1, nodes that are reachable in a given break will not appear in a greater one.

Details

If length(lim)>1, value is a list of length(from), containing lists of length(lim)

Value

List containing reachable nodes below cost limit(s).

Note

get_isochrone() recursively perform Dijkstra algorithm for each 'from' nodes and stop when cost limit is reached.

Examples

```
#Data describing edges of the graph
edges<-data.frame(from_vertex=c(0,0,1,1,2,2,3,4,4),
                  to_vertex=c(1,3,2,4,4,5,1,3,5),
                  cost=c(9,2,11,3,5,12,4,1,6))

#Construct directed graph
directed_graph<-makegraph(edges,directed=TRUE)

#Get nodes reachable around node 4 with maximum distances of 1 and 2
iso<-get_isochrone(Graph=directed_graph,from = "4",lim=c(1,2))

#With setdif set to TRUE
iso2<-get_isochrone(Graph=directed_graph,from = "4",lim=c(1,2),setdif=TRUE)
print(iso)
print(iso2)
```

---

get_multi_paths *Compute all shortest paths between origin and destination nodes.*

Description

Compute all shortest paths between origin and destination nodes.

Usage

```
get_multi_paths(Graph, from, to)
```

Arguments

Graph	An object generated by cppRouting::makegraph() function.
from	A vector of one or more vertices from which shortest paths are calculated (origin).
to	A vector of one or more vertices (destination).

Details

get_multi_paths() recursively perform Dijkstra algorithm for each 'from' nodes.

Value

List containing lists of shortest paths.

Note

Be aware that if 'from' and 'to' have consequent size, output will require much memory space.

Examples

```
#Data describing edges of the graph
edges<-data.frame(from_vertex=c(0,0,1,1,2,2,3,4,4),
                  to_vertex=c(1,3,2,4,4,5,1,3,5),
                  cost=c(9,2,11,3,5,12,4,1,6))

#Get all nodes
nodes<-unique(c(edges$from_vertex,edges$to_vertex))

#Construct directed and undirected graph
directed_graph<-makegraph(edges,directed=TRUE)
non_directed<-makegraph(edges,directed=FALSE)

#Get all shortest paths between all nodes in the two graphs
dir_paths<-get_multi_paths(Graph=directed_graph, from=nodes, to=nodes)
non_dir_paths<-get_multi_paths(Graph=non_directed, from=nodes, to=nodes)
print(dir_paths)
print(non_dir_paths)
```

`get_path_pair` *Compute shortest path between origin and destination nodes.*

Description

Compute shortest path between origin and destination nodes.

Usage

```
get_path_pair(Graph, from, to, algorithm = "Dijkstra", constant = 1)
```

Arguments

`Graph`	An object generated by cppRouting::makegraph() function.
`from`	A vector of one or more vertices from which shortest paths are calculated (origin).
`to`	A vector of one or more vertices (destination).
`algorithm`	character. "Dijkstra" for uni-directional Dijkstra, "bi" for bi-directional Dijkstra, "A*" or "NBA" for New bi-directional A star .Default to "Dijkstra"
`constant`	numeric. Constant to maintain the heuristic function admissible in A* algorithm. Default to 1, when cost is expressed in the same unit than coordinates. See details

Details

To perform A*, projected coordinates should be provided in the Graph object. In A* algorithms, euclidean distance is used as heuristic function. To understand how A star algorithm work, see https://en.wikipedia.org/wiki/A*_search_algorithm . To understand the importance of constant parameter, see the package description : https://github.com/vlarmet/cppRouting .

Value

List containing shortest path between from and to.

Note

'from' and 'to' must be the same length.

Examples

```
#Data describing edges of the graph
edges<-data.frame(from_vertex=c(0,0,1,1,2,2,3,4,4),
                  to_vertex=c(1,3,2,4,4,5,1,3,5),
                  cost=c(9,2,11,3,5,12,4,1,6))

#Get all nodes
nodes<-unique(c(edges$from_vertex,edges$to_vertex))

#Construct directed and undirected graph
directed_graph<-makegraph(edges,directed=TRUE)
non_directed<-makegraph(edges,directed=FALSE)

#Sampling origin and destination nodes
origin<-sample(nodes,10,replace=TRUE)
destination<-sample(nodes,10,replace=TRUE)

#Get distance between origin and destination in the two graphs
dir_paths<-get_path_pair(Graph=directed_graph, from=origin, to=destination)
non_dir_paths<-get_path_pair(Graph=non_directed, from=origin, to=destination)
print(dir_paths)
print(non_dir_paths)
```

---

makegraph *Construct graph*

Description

Construct graph

Usage

```
makegraph(df, directed = TRUE, coords = NULL)
```

Arguments

df	A data.frame or matrix containing 3 columns: from, to, cost. See details.
directed	logical. If FALSE, then all edges are duplicated by inverting 'from' and 'to' nodes.
coords	Optional. A data.frame or matrix containing all nodes coordinates. Columns order should be 'node_ID', 'X', 'Y'.

Details

'from' and 'to' are character or numeric vector containing nodes IDs. 'cost' is a non-negative numeric vector describing the cost (e.g time, distance) between each 'from' and 'to' nodes. coords should not be angles (e.g latitude and longitude), but expressed in a projection system.

Value

List

Examples

```
#Data describing edges of the graph
edges<-data.frame(from_vertex=c(0,0,1,1,2,2,3,4,4),
                  to_vertex=c(1,3,2,4,4,5,1,3,5),
                  cost=c(9,2,11,3,5,12,4,1,6))

#Construct directed and undirected graph
directed_graph<-makegraph(edges,directed=TRUE)
non_directed<-makegraph(edges,directed=FALSE)

#Visualizing directed and undirected graphs
if(requireNamespace("igraph",quietly = TRUE)){
  plot(igraph::graph_from_data_frame(edges))
  plot(igraph::graph_from_data_frame(edges,directed=FALSE))
}

#Coordinates of each nodes
coord<-data.frame(node=c(0,1,2,3,4,5),X=c(2,2,2,0,0,0),Y=c(0,2,2,0,2,4))

#Construct graph with coordinates
directed_graph2<-makegraph(edges, directed=TRUE, coords=coord)
```

Index