Package ‘cppRouting’

January 7, 2020

Type Package
Title Fast Implementation of Dijkstra Algorithm
Version 2.0
Date 2019-12-06
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>),
Contraction hierarchies (R. Geisberger, P. Sanders, D. Schultes and D. Delling (2008) <doi:10.1007/978-3-540-68552-4_24>),
License GPL (>= 2)
Imports Rcpp (>= 1.0.1), RcppParallel, RcppProgress, data.table
LinkingTo Rcpp, RcppParallel, RcppProgress
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 2020-01-07 17:00:10 UTC
cppRouting-package

**R topics documented:**

- cppRouting-package ............................................. 2
- cpp_contract .................................................. 3
- cpp_simplify ................................................... 4
- get_detour ..................................................... 5
- get_distance_matrix .......................................... 7
- get_distance_pair ............................................. 8
- get_isochrone .................................................. 10
- get_multi_paths ............................................... 11
- get_path_pair .................................................. 12
- makegraph ..................................................... 14
- to_df ............................................................ 15

---

**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. Implementation of contraction hierarchies 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,
- graph simplification by removing non intersection nodes, duplicated edges and isolated loops
- find nodes that can be reached under an additional detour time around the shortest path between two nodes
- contraction hierarchies

**Algorithms**

Algorithms that 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).
Algorithms that can be chosen for many-to-many calculations in get_distance_matrix() :

- many-to-many bidirectional search applied on a contracted graph,
- PHAST algorithm applied on a contracted graph.

References


cpp_contract

Description

Contract a graph by using contraction hierarchies algorithm

Usage

cpp_contract(Graph, silent = FALSE)

Arguments

Graph An object generated by makegraph() or cpp_simplify() function.
silent Logical. If TRUE, progress is not displayed.

Details

Contraction hierarchy is a speed-up technique for finding shortest path in a graph. It consist of two steps: preprocessing phase and query. cpp_contract() preprocess the input graph to later use special query algorithm implemented in get_distance_pair(), get_distance_matrix() and get_path_pair() functions. To see the benefits of using contraction hierarchies, see the package description: https://github.com/vlarmet/cppRouting/blob/master/README.md.

Value

A contracted graph.
cpp_simplify

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 cppRouting graph
graph <- makegraph(edges, directed=TRUE)

# Contract graph
contracted_graph <- cpp_contract(graph, silent=TRUE)

cpp_simplify

Reduce the number of edges by removing non-intersection nodes, duplicated edges and isolated loops in the graph.

Description

Reduce the number of edges by removing non-intersection nodes, duplicated edges and isolated loops in the graph.

Usage

cpp_simplify(Graph, keep = NULL, rm_loop = TRUE, iterate = FALSE,
silent = TRUE)

Arguments

Graph  An object generated by makegraph() function.
keep   Character or integer vector. Nodes of interest that will not be removed. Default to NULL
rm_loop Logical. if TRUE, isolated loops as removed. Default to TRUE
iterate Logical. If TRUE, process is repeated until only intersection nodes remain in the graph. Default to FALSE
silent Logical. If TRUE and iterate set to TRUE, number of iteration and number of removed nodes are printed to the console.

Details

To understand why process can be iterated, see the package description: https://github.com/vlarmet/cppRouting/blob/master/README.md

The first iteration usually eliminates the majority of non-intersection nodes and is therefore faster.

Value

The simplified cppRouting graph
Examples

#Simple directed graph
edges<-data.frame(from=c(1,2,3,4,5,6,7,8),
                  to=c(0,1,2,3,6,7,8,5),
                  dist=c(1,1,1,1,1,1,1,1))

#Plot
if(requireNamespace("igraph", quietly = TRUE)){
  igr<-igraph::graph_from_data_frame(edges)
  plot(igr)
}

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

#Simplify the graph, removing loop
simp<-cpp_simplify(graph, rm_loop=TRUE)

#Convert cppRouting graph to data frame
simp<-to_df(simp)

#Plot
if(requireNamespace("igraph", quietly = TRUE)){
  igr<-igraph::graph_from_data_frame(simp)
  plot(igr)
}

#Simplify the graph, keeping node 2 and keeping loop
simp<-cpp_simplify(graph, keep=2, rm_loop=FALSE)

#Convert cppRouting graph to data frame
simp<-to_df(simp)

#Plot
if(requireNamespace("igraph", quietly = TRUE)){
  igr<-igraph::graph_from_data_frame(simp)
  plot(igr)
}

get_detour

Return the nodes that can be reached in a detour time set around the shortest path

Description

Return the nodes that can be reached in a detour time set around the shortest path

Usage

get_detour(Graph, from, to, extra = NULL, keep = NULL, long = FALSE)
get_detour

Arguments

Graph An object generated by makegraph() or cpp_simplify() function.
from A vector of one or more vertices from which shortest path are calculated (origin).
to A vector of one or more vertices (destination).
extra numeric. Additional cost
keep numeric or character. Vertices of interest that will be returned.
long logical. If TRUE, a long data.frame is returned instead of a list.

Details
Each returned nodes $n$ meet the following condition:

$$SP(o,n) + SP(n,d) < SP(o,d) + t$$

with $SP$ shortest distance/time, $o$ the origin node, $d$ the destination node and $t$ the extra cost.

Modified bidirectional Dijkstra algorithm is ran for each path.

Value
List or a data.frame of nodes that can be reached

Note
’from’ and ’to’ must be the same length.

Examples

```r
if(requireNamespace("igraph", quietly = TRUE)){

#Generate fully connected graph
gf<- igraph::make_full_graph(400)
igraph::V(gf)$names<-1:400

#Convert to data frame and add random weights
df<-igraph::as_long_data_frame(gf)
df$dist<-sample(1:100,nrow(df),replace = TRUE)

#Construct cppRouting graph
graph<-makegraph(df[,c(1,2,5)],directed = FALSE)

#Pick up random origin and destination node
origin<-sample(1:400,1)
destination<-sample(1:400,1)

#Compute distance from origin to all nodes
or_to_all<-get_distance_matrix(graph,from=origin,to=1:400)

#Compute distance from all nodes to destination
all_to_dest<-get_distance_matrix(graph,from=1:400,to=destination)

#Get all shortest paths from origin to destination, passing by each node of the graph

```
get_distance_matrix

```r
    total_paths <- rowSums(cbind(t(or_to_all), all_to_dest))

    # Compute shortest path between origin and destination
    distance <- get_distance_pair(graph, from = origin, to = destination)

    # Compute detour with an additional cost of 3
    det <- get_detour(graph, from = origin, to = destination, extra = 3)

    # Check result validity
    length(unlist(det))
    length(total_paths[total_paths < distance + 3])
```

get_distance_matrix  Compute all shortest distance between origin and destination nodes.

**Description**

Compute all shortest distance between origin and destination nodes.

**Usage**

```r
get_distance_matrix(Graph, from, to, algorithm = "phast",
                     allcores = FALSE)
```

**Arguments**

- **Graph**: An object generated by makegraph(), cpp_simplify() or cpp_contract() 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. Only for contracted graph, "mch" for Many to many CH, "phast" for PHAST algorithm
- **allcores**: Logical. If TRUE, all cores are used.

**Details**

If graph is not contracted, `get_distance_matrix()` recursively perform Dijkstra algorithm for each 'from' nodes. If graph is contracted, the user has the choice between:

- many to many contraction hierarchies (mch) : optimal for square matrix.
- PHAST (phast) : outperform mch on rectangular matrix


**Value**

Matrix of shortest distances.
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 = "bi", constant = 1, allcores = FALSE)

Arguments

Graph An object generated by makegraph(), cpp_simplify() or cpp_contract() 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*" for A star unidirectional search or "NBA" for New bi-directional A star. Default to "Dijkstra"
constant numeric. Constant to maintain the heuristic function admissible in A* and NBA algorithms. Default to 1, when cost is expressed in the same unit than coordinates. See details
allcores Logical. If TRUE, all cores are used.
get_distance_pair

Details

If graph is not contracted, the user has the choice between:

- unidirectional Dijkstra (Dijkstra)
- A star (A*): projected coordinates should be provided
- bidirectional Dijkstra (bi)
- New bi-directional A star (NBA): projected coordinates should be provided

If the input graph has been contracted by cpp_contract() function, the algorithm is a modified bidirectional search.

In A* and New Bidirectional A star 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/blob/master/README.md

Value

Vector of shortest distances.

Note

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

Examples

```r
#Data describing edges of the graph
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, keep = NULL,
long = FALSE)

Arguments

Graph An object generated by makegraph() or cpp_simplify() 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.
keep numeric or character. Vertices of interest that will be returned.
long logical. If TRUE, a long data.frame is returned instead of a list.

Details

If length(lim)>1, value is a list of length(from), containing lists of length(lim). For large graph,
"keep" argument can be used for saving memory.

Value

List or a data.frame 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
directed_graph<-makegraph(edges,directed=TRUE)
get_multi_paths

#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, keep = NULL, long = FALSE)

Arguments

Graph  An object generated by makegraph() or cpp_simplify() 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).
keep    numeric or character. Vertices of interest that will be returned.
long    logical. If TRUE, a long data.frame is returned instead of a list.

Details

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

Value

List or a data.frame containing 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 = "bi", constant = 1,
              keep = NULL, long = FALSE)

Arguments

Graph       An object generated by makegraph(), cpp_simplify() or cpp_contract() 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*" for A star unidirectional search or "NBA" for New bi-directional A star.
             Default to "Dijkstra"
constant    numeric. Constant to maintain the heuristic function admissible in A* algorithm.
keep        numeric or character. Vertices of interest that will be returned.
long        logical. If TRUE, a long data.frame is returned instead of a list. Default to 1,
             when cost is expressed in the same unit than coordinates. See details
get_path_pair

Details

If graph is not contracted, the user has the choice between:

- unidirectional Dijkstra (Dijkstra)
- A star (A*): projected coordinates should be provided
- bidirectional Dijkstra (bi)
- New bi-directional A star (NBA): projected coordinates should be provided

If the input graph has been contracted by cpp_contract() function, the algorithm is a modified bi-directional search.

In A* and New Bidirectional A star 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/blob/master/README.md

Value

List or a data.frame containing shortest path nodes between from and to.

Note

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

Examples

#Data describing edges of the graph
data.frame(from_vertex=c(0,0,1,2,1,3,4,4,4),
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
unique(c(data.frame(from_vertex)),data.frame(to_vertex))

#Construct directed and undirected graph
directed_graph<-makegraph(data.frame(edges,directed=TRUE)
non_directed<-makegraph(data.frame(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)
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 with two useful attributes for the user:

nbnode : total number of vertices
dict$ref : vertices IDs

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)

to_df

## to_df

### Convert cppRouting graph to data.frame

**Description**
Convert cppRouting graph to data.frame

**Usage**
to_df(Graph)

**Arguments**
- `Graph`: An object generated by cppRouting::makegraph() or cpp_simplify() function.

**Value**
Data.frame with from, to and dist column

**Examples**

# Simple directed graph
edges <- data.frame(from = c(1, 2, 3, 4, 5, 6, 7, 8),
                    to = c(0, 1, 2, 3, 6, 7, 8, 5),
                    dist = c(1, 1, 1, 1, 1, 1, 1, 1))

directed_graph <- makegraph(edges, directed = TRUE)

to_df(directed_graph)
Index

cpp_contract, 3
cpp_simplify, 4
cppRouting (cppRouting-package), 2
cppRouting-package, 2

get_detour, 5
get_distance_matrix, 7
get_distance_pair, 8
get_isochrone, 10
get_multi_paths, 11
get_path_pair, 12

makegraph, 14

to_df, 15