Package ‘netgen’

January 8, 2020

Type   Package
Title  Network Generator for Combinatorial Graph Problems
Description  Methods for the generation of a wide range of network geographies,
e.g., grid networks or clustered networks. Useful for the generation of
benchmarking instances for the investigation of, e.g., Vehicle-Routing-Problems
or Travelling Salesperson Problems.
Version  1.3.2
Date    2020-01-08
Maintainer  Jakob Bossek <j.bossek@gmail.com>
URL     https://github.com/jakobbossek/netgen
BugReports  https://github.com/jakobbossek/netgen/issues
License  BSD_3_clause + file LICENSE
Depends  BBmisc (>= 1.6), mvtnorm (>= 1.0-2), lhs (>= 0.10), checkmate
        (>= 1.8.0)
Imports  ggplot2, igraph (>= 0.7.1), stringr (>= 0.6.2)
Suggests  testthat, lpSolve
LazyData  yes
ByteCompile  yes
RoxygenNote  6.1.1
NeedsCompilation  no
Author  Jakob Bossek [aut, cre]
Repository  CRAN
Date/Publication  2020-01-08 17:50:02 UTC

R topics documented:

  addNodeWeights .......................................................... 2
  as.character.Network .................................................... 3
  as.data.frame.Network .................................................... 3

1
addNodeWeights

Add node weights.

Description

This function adds node weights to an edge-weighted graph. This is of interest in the study of weighted TSP, where the distance between two nodes is not based on the actual distance but additionally is influenced by the weight of the starting node or all nodes prior in the permutation. This is indeed of practical interest. E.g. consider a garbage collecting vehicle which requires the more gas per mile the heavier its load.

Usage

addNodeWeights(x, weights = NULL)
as.character.Network

Arguments

x  [Network]  Network.
weights  [numeric(1)]  Numeric vector of weights.

See Also  
generateRandomNetwork, generateClusteredNetwork, generateGridNetwork

as.character.Network  Get basic network information as a string.

Description  
Get basic network information as a string.

Usage

## S3 method for class 'Network'
as.character(x, ...)

Arguments

x  [Network]  Network.
...
[any]  Not used at the moment.

Value

character(1)

as.data.frame.Network  Convert network to data frame.

Description  
Convert network to data frame.

Usage

## S3 method for class 'Network'
as.data.frame(x, row.names = NULL, optional = FALSE,
  include.extras = TRUE, ...)
Argument

\texttt{x} \hspace{1cm} \text{[Network]} \hspace{1cm} \text{Network.}

\texttt{row.names} \hspace{1cm} \text{[character]} \hspace{1cm} \text{Row names for the result. Default is \texttt{NULL}.}

\texttt{optional} \hspace{1cm} \text{[any]} \hspace{1cm} \text{Currently not used.}

\texttt{include.extras} \hspace{1cm} \text{[logical(1)]} \hspace{1cm} \text{Include additional information like cluster membership and node type as specific columns? Default is \texttt{TRUE}.}

... \hspace{1cm} \text{[any]} \hspace{1cm} \text{Currently not used.}

Value

data.frame

Note

If the instance contains of \(n\) depots, the depot coordinates fill the first \(n\) rows of the data frame.

\begin{center}
\begin{tabular}{ll}
\textbf{as.matrix.Network} & \textit{Convert network to matrix.} \\
\end{tabular}
\end{center}

Description

Convert network to matrix.

Usage

\begin{verbatim}
## S3 method for class 'Network'
as.matrix(x, ...)
\end{verbatim}

Arguments

\begin{verbatim}
\texttt{x} \hspace{1cm} \text{[Network]} \hspace{1cm} \text{Network.}
\texttt{...} \hspace{1cm} \text{[any]} \hspace{1cm} \text{Currently not used.}
\end{verbatim}

Value

\begin{verbatim}
matrix
\end{verbatim}

Note

If the instance contains of \(n\) depots, the depot coordinates fill the first \(n\) rows of the matrix.
Description

Generates a ggplot object. Nice possibility to visualize 2-dimensional (clustered) networks in the euclidean plane.

Usage

```r
## S3 method for class 'Network'
autoplot(object, path = NULL, close.path = FALSE,
         path.colour = "gray", use.opt.tour = FALSE, ...)
```

Arguments

- `object` [Network], Network.
- `path` [integer], An integer vector containing the order of cities of a path or a list of multiple paths. Keep in mind that instances with \( n \) nodes and \( m \) depots have \( n + m \) coordinates, with the 1,...,\( m \) first coordinates belonging to the depots.
- `close.path` [logical(1)], Logical indicating whether the path passed by `path` should be closed to a cycle. Default is FALSE.
- `path.colour` [character(1)], Colour of the lines linking nodes on a path. Default is “gray”.
- `use.opt.tour` [logical(1)], If the given network knows its optimal tour, should it be plotted? If this is the case and `path` is given additionally, the optimal tour is ignored. Default is FALSE.
- `...` [any], Currently not used.

Value

ggplot

Examples

```r
## Not run:
# here we have no depots ...  
x = generateClusteredNetwork(n.points = 30L, n.cluster = 2L)  
pl = autoplot(x, path = 1:3)  
# ... and here we have two depots: the path visits the depots in this case  
x = generateRandomNetwork(n.points = 30L, n.depots = 2L)
```
dynamise

Add dynamic arrival times to nodes.

Description

Some variants of the Vehicle Routing Problem (VRP) consider static as well as dynamic customers (nodes). This function takes a Network and dynamises it, i.e., it adds dynamic arrival times to the customers via a Poisson process.

Usage

dynamise(x, n.dynamic = NULL, dyn.customers.ratio = NULL, arrival.limit)

Arguments

x [Network] Network.
n.dynamic [integer(1) | NULL] Number of nodes, which should become dynamic. Ignored if dyn.customers.ratio is not NULL.
dyn.customers.ratio [numeric(1) | NULL] Ratio of dynamic nodes. If this is set to a numeric value in (0, 1), the parameter n.dynamic is ignored.
arrival.limit [numeric(1)] Maximal arrival time.

Value

Network Modified network (now has an additional list element 'arrival.times') and the ratio of dynamic customers as an attribute.

See Also

generateRandomNetwork, generateClusteredNetwork, generateGridNetwork

Examples

x = generateClusteredNetwork(n.points = 100L, n.cluster = 4L, upper = 100, n.depots = 2L)
x = dynamise(x, dyn.customers.ratio = 0.3, arrival.limit = 400)
print(x)
**exportToFile**

Exports a network to a proprietary format.

**Description**

The format used is similar to the TSPlib format (see `exportToTSPlibFormat`), but it saves not only the point coordinates. It also saves the arrival times of dynamic customers.

**Usage**

```r
exportToFile(x, filename, digits = 2L)
```

**Arguments**

- `x`: [Network]
  - Network to export.
- `filename`: [character(1)]
  - File name.
- `digits`: [integer(1)]
  - Round coordinates to this number of digits. Default is 2.

**Value**

Nothing

---

**exportToTSPlibFormat**

Exports a network to the TSPlib format.

**Description**

Exports a network to the TSPlib format.

**Usage**

```r
exportToTSPlibFormat(x, filename, name = NULL, comment = NULL,
                       use.extended.format = TRUE, full.matrix = FALSE, digits = 10L)
```

**Arguments**

- `x`: [Network]
  - Network to export.
- `filename`: [character(1)]
  - File name.
name [character(1) | NULL]
Character string describing the instance. Used for the NAME field in the TSPlib file format. Otherwise, the name of the instance is used. If the latter is NULL, this parameter is mandatory.

comment [character(1) | NULL]
Optional string with additional information about the instance. Used for the COMMENT field. If not provided the comment field of the instance is used. If the latter is NULL, no comment at all is saved.

use.extended.format [logical(1)]
Use the “extended tsplib format” with additional information like cluster membership and bounds? Default is TRUE.

full.matrix [logical(1)]
Make use of “FULL\_MATRIX” “EDGE\_WEIGHT\_FORMAT” instead of node coordinates? Default is FALSE.

digits [integer(1)]
Round coordinates to this number of digits. Default is 10.

Value
Nothing

Note
Currently we only support euclidean 2D instances. Furthermore note, that if use.extended.format is TRUE, most alternative TSPlib parsers will most probably not be able to parse the generated file.

filterTSPInstances
Filter TSPlib instances according to its specifications.

Description
Given a directory of TSP problems in the TSPlib format with file extension .tsp, this function reads the specifications of each TSPlib instance in that directory and returns a data frame with rowwise information about each instance.

Basically the function is a wrapper around getTSPInstancesOverview.

Usage
filterTSPInstances(directory = NULL, expr, paths.only = FALSE, opt.known = FALSE)
generateClusteredNetwork

Function for generation of clustered networks

Description

This function generates clustered networks. It first generates \( n \) cluster centers via a latin hypercube design to ensure space-filling property, i. e., to ensure, that the clusters are placed far from each other. It then distributes points to the clusters according to gaussian distributions using the cluster centers as the mean vector and the distance to the nearest neighbour cluster center as the variance. This procedure works well if the box constraints of the hypercube are not too low (see the lower bound for the upper parameter).

Arguments

directory [character(1)]
Readable directory path.

expr [expression]
Expression wrapped with the quote function.

paths.only [logical(1)]
Should only the full file names of the instances be returned? Default is FALSE.

opt.known [logical(1)]
Filter instances \( x \) with unknown optimal tour length (given in file \( x \).tsp.tour)? Default is FALSE.

Value
data.frame

See Also

getTSPInstancesOverview

Examples

```r
## Not run:
# Get a data frame of instances and its properties for all instances
# with more than 4000 nodes
filterTSPInstances("path/to/instances", quote(dimension > 4000))

# Now get only the full file names of all instances with edge weight type
# EUC_2D or CEIL_2D (see tsplib documentation for details)
filterTSPInstances("path/to/instances",
  expr = quote(edge_weight_type %in% c("EUC_2D", "CEIL_2D")),
  paths.only = TRUE
)
## End(Not run)
```
Usage

generateClusteredNetwork(n.cluster, n.points, n.dim = 2L,
generator = lhs::maximinLHS, lower = 0, upper = 100,
sigmas = NULL, n.depots = NULL,
distribution.strategy = "equally.distributed",
cluster.centers = NULL, out.of.bounds_handling = "mirror",
name = NULL, ...)

Arguments

n.cluster [integer(1)]
Desired number of clusters. This is ignored if cluster.centers is provided.
n.points [integer(1)]
Number of points for the network.
n.dim [integer(1)]
Number of dimensions. Default ist 2.
generator [function]
Function which generates cluster centers. Default is maximinLHS.
lower [numeric(1)]
Lower box constraint for cube. Default is 0.
upper [numeric(1)]
Upper box constraint for cube. Default is 100.
sigmas [list | NULL]
Unnamed list of length n.cluster containing a covariance matrix for each cluster. Default is NULL. In this case the covariance matrix is a diagonal matrix containing the distances to the nearest neighbour cluster center as diagonal elements.
n.depots [integer(1)]
Number of depots in instances for the Vehicle Routing Problem (VRP). Default is NULL, i.e., no depots. The proceeding is as follows: If n.depots is 1L, a random cluster center is defined to be the depot. If n.depots is 2L, the second depot has maximal distance to the first. At the moment at most two depots are possible.
distribution.strategy [character(1)]
Define the strategy to distribute n.points on the n.cluster clusters. Default is “equally.distributed”, which is the only option at the moment.
cluster.centers [matrix]
Matrix of cluster centres of dimension n.cluster x n.dim. If this is set, cluster centres are not generated automatically. Default is NULL.
out.of.bounds_handling [character(1)]
Clusters are generated on base of a multivariate gaussian distribution with the cluster center as the mean vector. Possibly some of the points might fall out of
generateGridNetwork

Generates a grid network.

Description

Generates a grid network.

Usage

generateGridNetwork(n.points.per.dim = NULL, n.dim = 2L, lower = 0L, upper = 100L, name = NULL)

Arguments

n.points.per.dim [integer(1)]
Number of points in each dimension.

n.dim [integer(1)]
Number of dimensions. Default is 2.

lower [numeric(1)]
Lower box constraint for cube. Default is 0.

Value

ClusteredNetwork Object of type ClusteredNetwork.

Examples

x = generateClusteredNetwork(n.points = 20L, n.cluster = 2L)
y = generateClusteredNetwork(n.points = 40L, n.cluster = 3L, n.depots = 2L)
z = generateClusteredNetwork(n.points = 200L, n.cluster = 10L, out.of.bounds.handling = "reset")
generateRandomNetwork

upper [numeric(1)]
Upper box constraint for cube. Default is 100.

name [character(1) | NULL]
Optional name for the generated network. Default is NULL. In this case a random name is generated.

Value
Network

Note
Grid networks with depots are not supported at the moment.

Examples
x = generateGridNetwork(n.points.per.dim = 10L, upper = 50)

generateRandomNetwork  Generates a random graph in a hypercube.

Description
Generates a random graph in a hypercube.

Usage
generateRandomNetwork(n.points, n.dim = 2L, n.depots = NULL,
lower = 0, upper = 100, name = NULL)

Arguments
n.points [integer(1)]
Number of points.

n.dim [integer(1)]
Number of dimensions. Default is 2.

n.depots [integer(1)]
Number of depots in instances for the Vehicle Routing Problem (VRP). Default is NULL, i.e., no depots. The proceeding is as follows: If n.depots is 1, a random cluster center is defined to be the depot. If n.depots is 2, the second depot has maximal distance to the first. By convention the depots are placed as the first nodes in the coordinates matrix.

lower [numeric(1)]
Lower box constraint of cube.

upper [numeric(1)]
Upper box constraint of cube. Default is 100.
getDepotCoordinates

name [character(1) | NULL]
Optional name for the generated network. Default is NULL. In this case a random name is generated.

Value
Network

Examples
x = generateRandomNetwork(n.points = 100L, n.depots = 2L, upper = 50)

---

generateRandomNetwork

Description
Get coordinates of depots.

Usage
generateRandomNetwork(x)

Arguments
x [Network]
Network.

Value
matrix

---

generateRandomNetwork

Description
Get the number of clusters of a network.

Usage
generateRandomNetwork(x)

Arguments
x [Network]
Network.
getNumberOfNodes

Value
integer(1) Number of clusters.

Note
For simple random or grid networks this function always returns 1.

getNumberOfDepots

Description
Returns the number of depots of a network.

Usage
getNumberOfDepots(x)

Arguments
x [Network]
    Network.

Value
integer(1)

getNumberOfNodes

Description
Returns number of nodes of a network.

Usage
getNumberOfNodes(x)

Arguments
x [Network]
    Network.

Value
integer(1) Number of nodes of the network.
getOptimalPointMatching

Computes optimal point assignment for two sets of points of equal size.

Description

Internally it handles the points and the possible matchings as a bi-partite graphs and finds an optimal matching due to euclidean distance by an efficient linear programming solver.

Usage

getOptimalPointMatching(x, y, method = "lp", full.output = FALSE)

Arguments

x  
[Network | matrix]
First network or matrix of coordinates of the first point set.

y  
[Network | matrix]
Second network or matrix of coordinates of the second point set.

method  
[character(1)]
Method used to solve the assignment problem. There are currently two methods available:

lp  Solves the problem be means of linear programming with the lpSolve package to optimality. This is the default.

push_relabel  The assignment problem can be formulated as a matching problem on bipartite graphs. This method makes use of the push-relabel algorithm from theigraph. Solves to optimality.

random  Random point matching. Just for comparison.

greedy  Greedy point matching, i.e., iteratively assign two unmatched points with minimal euclidean distance.

full.output  
[logical(1)]
Should optimization process information, e.g., the weight of the best matching, be returned? Default is FALSE.

Value

matrix | list  Either a matrix where each row consists of the indices of the pairwise assigned points. If full.output = TRUE a list is returned with the assignment matrix "pm", the method "method" and the optimal weight "opt.weight".

See Also

visualizePointMatching
getPointDistributionStrategies

*Returns the available strategies for distributing points around clusters.*

### Description

Returns the available strategies for distributing points around clusters.

### Usage

```r
getCodePointDistributionStrategies()
```

### Value

character

getTSPInstancesOverview

*Get an overview of instances in a directory.*

### Description

This function expects a directory and returns a data frame containing the most important properties, e.g., dimension, edge weight type, of all TSPlib instances (with file extensions tsp) in that directory. Moreover, the data frame contains information on the availability of the optimal tour length (files tsp.opt) and optimal tour (tsp.tour).

### Usage

```r
getCodeTSPInstancesOverview(directory, append.filename = FALSE)
```

### Arguments

- **directory** [character(1)]
  
  Readable directory path.

- **append.filename** [logical(1)]
  
  Should the full file names be appended to the data frame? Default is FALSE.

### Value

data.frame
getValidEdgeWeightTypes

*Get TSPlib edge weight types.*

**Description**

Get TSPlib edge weight types.

**Usage**

getValidEdgeWeightTypes()

---

hasDepots

*Check if network has depots.*

**Description**

Check if network has depots.

**Usage**

hasDepots(x)

**Arguments**

x  
[Network]  
Network.

**Value**

logical(1)
importFromFile

*Import a network from proprietary format.*

**Description**

Import a network from proprietary format.

**Usage**

```
importFromFile(filename)
```

**Arguments**

- `filename` [character(1)]
  File name.

**Value**

Nothing

---

importFromTSPlibFormat

*Import network from (extended) TSPlib format.*

**Description**

Import network from (extended) TSPlib format.

**Usage**

```
importFromTSPlibFormat(filename, round.distances = TRUE, read.opt = TRUE)
```

**Arguments**

- `filename` [character(1)]
  Path to TSPlib file.

- `round.distances` [logical(1)]
  Should the distances of EUC_2D instances be rounded to the nearest integer value? Default is TRUE.

- `read.opt` [logical(1)]
  Should the optimal tour length (in file filename.opt) and the optimal tour (in file filename.tour) be loaded if available? Default is TRUE.
isEuclidean

Value
Network  Network object.

Note
The extended TSPlib contains additional specification parts and a cluster membership section. Currently only the import of symmetric TSP instances is possible.

isEuclidean  Check if network is euclidean.

Description
Check if a Network object has euclidean coordinates.

Usage
isEuclidean(x)

Arguments
x  [Network]
    Network.

Value
logical(1)

isNetwork  Check if object is Network.

Description
Check if object is Network.

Usage
isNetwork(x)

Arguments
x  [any]
    Arbitrary R object.

Value
logical(1)
**makeNetwork**

*Generate network based on coordinates.*

**Description**

Create a (clustered) network object.

**Usage**

```r
makeNetwork(coordinates, distance.matrix = NULL, name = NULL,
             comment = NULL, membership = NULL, edge.weight.type = NULL,
             depot.coordinates = NULL, lower = NULL, upper = NULL,
             opt.tour.length = NULL, opt.tour = NULL, node.weights = NULL)
```

**Arguments**

- **coordinates** [matrix]
  - Numeric matrix of 2D coordinates.
- **distance.matrix** [matrix]
  - Optional distance matrix.
- **name** [character(1) | NULL]
  - Optional name of the network.
- **comment** [character | NULL]
  - Optional additional comments on instance.
- **membership** [numeric | NULL]
  - Optional vector of memberships for clustered networks.
- **edge.weight.type** [character(1) | NULL]
  - The edge weight type indicates how edge weights are represented in the TSPlib format. If `distance.matrix` is NULL, the passed value is ignored and EUC\_2D is assigned. Otherwise the edge weight type must be one of the following {EUC\_2D,EUC\_3D,MAX\_2D,MAX\_3D,MAN\_2D,MAN\_3D,CEIL\_2D,GEO,ATT,EXPLICIT}.
- **depot.coordinates** [matrix | NULL]
  - Numeric matrix of 2D coordinates of depots. Default is NULL, which means no depots at all.
- **lower** [numeric(1)]
  - Lower box constraint of cube.
- **upper** [numeric(1)]
  - Upper box constraint of cube.
- **opt.tour.length** [numeric(1)]
  - Optional length of the optimal roundtrip tour. Default is NULL, which means the tour length is unknown.
morphInstances

opt.tour [integer]
Optional optimal permutation of node indicies. Default is NULL, which means the optimal tour is unknown.

node.weights [numeric]
Vector of node weights (for weighted version of TSP). Default is NULL, i.e., no node weights at all.

Value

Network

Description

This function takes two (clustered) networks with equal number of nodes and, if present, equal number of depots, and generates another instance by applying a convex combination to the coordinates of node pairs. The node pairs are determined by a point matching algorithm, which solves this assignment problem via a integer programming procedure. If both instances contain depots, point matching is done separately on depots and the remaining nodes.

Usage

morphInstances(x, y, alpha, point.matching = NULL, point.matching.algorithm = "push_relabel")

Arguments

x [Network]
First network.

y [Network]
Second network.

alpha [numeric(1)]
Coefficient alpha for convex combination.

point.matching [matrix|NULL]
Point matching which shall be used for morphing. If NULL, an optimal point matching is generated via function getOptimalPointMatching. Default is NULL. Currently it is just possible to pass a point matching for instances without depots.

point.matching.algorithm [function]
Algorithm used to find a point matching. . See argument method of getOptimalPointMatching.

Value

Network Morphed network.
rescaleNetwork

Rescale network

Description

Normalize network coordinates to the unit cube while maintaining its geography.

Usage

rescaleNetwork(x, method = "global2")

Arguments

x
[Network]
Network.

method
[character(1)]
Rescaling method which actually modifies the coordinates. Currently there are three methods available:

by.dimension Scaling is performed for each dimension independently.

global Here we shift all the points toward the origin by the minimum of both x and y coordinate and divide by the range of global maximum and minimum.

global2 Here we shift - analogously to the by.dimension strategy - dimension wise and divide by the maximum of the ranges in x respectively y direction. Default is global2, which leads to the most “natural” rescaling.

Value

Network

See Also

visualizeMorphing, visualizePointMatching

Examples

x = generateRandomNetwork(n.points = 40L, n.depots = 2L)
y = generateClusteredNetwork(n.points = 40L, n.cluster = 2L, n.depots = 2L)
z = morphInstances(x, y, alpha = 0.2, point.matching.algorithm = "push_relabel")
## Not run:
library(gridExtra)
plot.list = list(autoplot(x), autoplot(z), autoplot(y))
plot.list$nrow = 1
do.call(grid.arrange, plot.list)
## End(Not run)
visualizeMorphing

Examples

## Not run:
library(gridExtra)
x = generateClusteredNetwork(n.points = 100L, n.cluster = 4L, name = "Rescaling Demo")

# here we "stretch" the instance x direction to visualize the differences of
# the rescaling methods
x$coordinates[, 1] = x$coordinates[, 1] * 10L
x$upper = x$upper * 10L
pls = list(
    autoplot(x) + ggtitle("Original"),
    autoplot(rescaleNetwork(x, method = "by.dimension")) + ggtitle("By dimension"),
    autoplot(rescaleNetwork(x, method = "global")) + ggtitle("Global"),
    autoplot(rescaleNetwork(x, method = "global2")) + ggtitle("Global2")
)
pls$nrow = 1L
do.call(grid.arrange, pls)

## End(Not run)

visualizeMorphing     Fancy visualization of morphing.

Description

Takes two instances of equal size and some alpha values. Computes the point matching and mor-
phings for the alpha values and visualizes the transition of points of the first instance towards their
matched counterparts of the second instance with two different methods.

Usage

visualizeMorphing(x, y, point.matching = NULL, alphas = c(0.25, 0.5,
0.75), arrows = TRUE, in.one.plot = TRUE, point.colour = NULL)

Arguments

x     [Network]
First network.

y     [Network]
Second network.

point.matching     [matrix]
Point matching which shall be used for morphing. If NULL, an optimal point
matching is generated via function getOptimalPointMatching. Default is NULL.

alphas    [numeric]
Vector of coefficients ‘alpha’ for convex combinations.

arrows    [logical(1)]
Draw arrows originating in the points of x and ending in the points matched in
y. Default is TRUE.
in.one.plot [logical(1)]
Currently the function offers two different types of plot. If in.one.plot is
TRUE, which is the default value, the morphing is depicted in one plot. This
is in particular useful for small instances. If set to FALSE, a matrix of plots is
generated via facet_grid. One plot for each alpha value in alphas.

point.colour [character(1)]
Which colour to use for the non-depot points? Default is NULL. In this case the
points are coloured by membership. Only considered if in.one.plot is FALSE.

Value

ggplot

See Also

morphInstances

visualizePointMatching

Visualize point matching.

Description

Visualize a point matchings. Points and lines between the matched points are drawn in order to
visualize the assignment.

Usage

visualizePointMatching(x, y, point.matching, highlight.longest = 0L)

Arguments

x [Network | matrix]
Network or (n x 2) matrix.

y [Network | matrix]
Network or (n x 2) matrix.

point.matching [matrix]
Point matching received via getOptimalPointMatching for example.

highlight.longest [integer(1)]
Number of longest distances which should be particularly highlighted. Default
is 0.

Value

ggplot
**visualizePointMatching**

See Also

getOptimalPointMatching, morphInstances, visualizeMorphing

Examples

```r
# point matching on networks
x = generateRandomNetwork(n.points = 20L, upper = 100)
y = generateClusteredNetwork(n.points = 20L, n.cluster = 2L, upper = 100)
## Not run:
pm = getOptimalPointMatching(x$coordinates, y$coordinates)
print(visualizePointMatching(x, y, pm, highlight.longest = 2L))
## End(Not run)

# point matching on point clouds
x = matrix(runif(20L), 2L)
y = matrix(runif(20L), 2L)
## Not run:
pm = getOptimalPointMatching(x, y)
print(visualizePointMatching(x, y, pm))
## End(Not run)
```
Index

addNodeWeights, 2
as.character.Network, 3
as.data.frame.Network, 3
as.matrix.Network, 4
autoplot.Network, 5
dynamise, 6
exportToFile, 7
exportToTSPlibFormat, 7, 7
facet_grid, 24
filterTSPInstances, 8
generateClusteredNetwork, 3, 6, 9
generateGridNetwork, 3, 6, 11
generateRandomNetwork, 3, 6, 11, 12
getDepotCoordinates, 13
getNumberOfClusters, 13
getNumberOfDepots, 14
getNumberOfNodes, 14
getOptimalPointMatching, 15, 21, 23, 25
getPointDistributionStrategies, 16
getTSPInstancesOverview, 8, 9, 16
getValidEdgeWeightTypes, 17
ggplot, 5, 24
hasDepots, 17
importFromFile, 18
importFromTSPlibFormat, 18
isEuclidean, 19
isNetwork, 19
makeNetwork, 20
maximinLHS, 10
morphInstances, 21, 24, 25
rescaleNetwork, 22
visualizeMorphing, 22, 23, 25
visualizePointMatching, 15, 22, 24