Package ‘netgen’

January 22, 2016

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
Date 2016-01-21
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 (>= 1.5.1)
Imports ggplot2, lpSolve, igraph (>= 0.7.1), stringr (>= 0.6.2)
Suggests testthat
LazyData yes
ByteCompile yes
RoxygenNote 5.0.1
NeedsCompilation no
Author Jakob Bossek [aut, cre]
Repository CRAN
Date/Publication 2016-01-22 10:26:50

R topics documented:

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

1
Quote variables to create a list of unevaluated expressions for later evaluation.

Description

This function is used by filterTSPInstances to pass unevaluated expressions.

Usage

\(.\ldots\)

Arguments

\ldots [any] Unevaluated expressions to be recorded.

Value

List of symbol and language primitives.
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, ...)

Arguments

x [Network]
Network.

row.names [character]
Row names for the result. Default is NULL.

optional [any]
Currently not used.
**include.extras** [logical(1)]
Include additional information like cluster membership and node type as specific columns? Default is TRUE.

... [any]
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.

**Description**
Convert network to matrix.

**Usage**

```r
## S3 method for class 'Network'
as.matrix(x, ...)
```

**Arguments**

- **x** [Network]
  Network.

- ... [any]
  Currently not used.

**Value**
matrix

**Note**
If the instance contains of \( n \) depots, the depot coordinates fill the first \( n \) rows of the matrix.
autoplot.Network  

Autoplot function.

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", ...)
```

Arguments

- `object` [Network]
  - Network.
- `path` [integer]
  - An integer vector containing the order of cities of a path. Keep in mind, that instances with \( n \) nodes and \( m \) depots have \( n + m \) coordinates, with the \( 1, \ldots, m \) 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”.
- `...` [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)
pl = autoplot(x, path = 1:3, path.colour = "tomato")
```

## End(Not run)
**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**

```r
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**

```r
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 an 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

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

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.
filterTSPInstances

Description
Given a directory, this function reads the specifications of each TSPlib instance in that directory and returns a subset.

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

Arguments

directory [character(1)]
Readable directory path.

eexpr [expression]
Expression wrapped with the . function.

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

**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", 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 = edge_weight_type %in% c("EUC_2D", "CEIL_2D"),
  paths.only = TRUE
)

## End(Not run)
```

---

**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).

**Usage**

```r
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 is 2.

generator [function]
Function which generates cluster centers. Default is maximin.LHS.

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

upper [numeric(1)]
Upper box constant 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 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. 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 bounds, i.e., get coordinates larger than upper or lower than lower. There are two strategies to force them to stick to the bounds:

“reset” Set the violating coordinates to the bounds.
“mirror” Mirror the coordinates at the violated axis.
Default is “mirror”.

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

...[any]
Currently not used.

Value
ClusteredNetwork Object of type ClusteredNetwork.

See Also
generateRandomNetwork

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")

generateGridNetwork Generates a grid network.

Description
Generates a grid network.

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

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

\[
x = \text{generateGridNetwork}(n.\text{points}.\text{per.dim} = 10L, \text{upper} = 50)
\]

\[
\text{generateRandomNetwork} \quad \text{Generates a random graph in a hypercube.}
\]

Description

Generates a random graph in a hypercube.

Usage

\[
\text{generateRandomNetwork}(n.\text{points}, n.\text{dim} = 2L, n.\text{depots} = \text{NULL}, \text{lower} = 0, \text{upper} = 100, \text{name} = \text{NULL})
\]

Arguments

\[
\begin{align*}
\text{n.points} & : [\text{integer}(1)] \\
& \quad \text{Number of points.} \\
\text{n.dim} & : [\text{integer}(1)] \\
& \quad \text{Number of dimensions. Default ist 2.} \\
\text{n.depots} & : [\text{integer}(1)] \\
& \quad \text{Number of depots in instances for the Vehicle Routing Problem (VRP). Default is NULL, i. e., no depots. The proceeding is as follows: If n.depots is 1, a random cluster center is defined to be the depot. If n.depots is 2, the second depot has maximal distance to the first. By convention the depots are placed as the first nodes in the coordinates matrix.} \\
\text{lower} & : [\text{numeric}(1)] \\
& \quad \text{Lower box constraint of cube.} \\
\text{upper} & : [\text{numeric}(1)] \\
& \quad \text{Upper box constraint of cube. Default is 100.} \\
\text{name} & : [\text{character}(1) | \text{NULL}] \\
& \quad \text{Optional name for the generated network. Default is NULL. In this case a random name is generated.}
\end{align*}
\]

Value

Network

Examples

\[
x = \text{generateRandomNetwork}(n.\text{points} = 100L, n.\text{depots} = 2L, \text{upper} = 50)
\]
**getDepotCoordinates**

*Get coordinates of depots.*

**Description**

Get coordinates of depots.

**Usage**

`getDepotCoordinates(x)`

**Arguments**

`x` [Network]

Network.

**Value**

Matrix

---

**getNumberOfClusters**

*Get the number of clusters of a network.*

**Description**

Get the number of clusters of a network.

**Usage**

`getNumberOfClusters(x)`

**Arguments**

`x` [Network]

Network.

**Value**

`integer(1)` Number of clusters.

**Note**

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

*Returns the number of depots of a network.*

**Description**

Returns the number of depots of a network.

**Usage**

`getNumberOfDepots(x)`

**Arguments**

- `x`  
  `[Network]`  
  Network.

**Value**

`integer(1)`

getNumberOfNodes

*Returns number of nodes of a network.*

**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")

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. 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 the igraph.

random Random point matching.

Value

matrix Each row consists of the indizes of the pairwise matchings.

See Also

visualizePointMatching
getPointDistributionStrategies

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

**Description**

Returns the available strategies for distributing points around clusters.

**Usage**

getPointDistributionStrategies()

**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 in that directory.

**Usage**

getTSPInstancesOverview(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**

```plaintext
getValidEdgeWeightTypes()
```

---

hasDepots

Check if network has depots.

**Description**

Check if network has depots.

**Usage**

```plaintext
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)
```

**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.

**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
makeNetwork(coordinates, distance.matrix = NULL, name = NULL, 
            comment = NULL, membership = NULL, edge.weight.type = NULL, 
            depot.coordinates = NULL, lower = NULL, upper = 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, EX}.

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.

Value
Network
### morphInstances

**Morphing of two networks with a convex combination of the coordinates.**

#### 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

```r
morphInstances(x, y, alpha, point.matching = NULL,
              point.matching.algorithm = getoptimalpointmatching)
```

#### 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. Default is `getOptimalPointMatching`.

#### Value

Network  Morphed network

#### See Also

- `visualizeMorphing`, `visualizePointMatching`

#### Examples

```r
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)
```

## 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)

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 coordinates 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 respectivly y direction. Default is global2, which leads to the most “natural” rescaling.

Value

Network

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(
visualizeMorphing

Fancy visualization of morphing.

Description
Takes two instances of equal size and some alpha values. Computes the point matching and morphings 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)

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 dipicted 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.

Value
ggplot
**visualizePointMatching**

*Visualize point matching.*

**Description**

Draw the points and lines between the matched points for visualization.

**Usage**

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

**Arguments**

- `x` [Network]
  - First network.
- `y` [Network]
  - Second network.
- `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`

**See Also**

`getOptimalPointMatching`, `morphInstances`, `visualizeMorphing`

**Examples**

```r
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)
```
Index

.., 2, 8

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, 23
filterTSPInstances, 8
generateClusteredNetwork, 6, 9
generateGridNetwork, 6, 11
generateRandomNetwork, 6, 11, 12
getDepotCoordinates, 13
getNumberOfClusters, 13
getNumberOfDepots, 14
getNumberOfNodes, 14
getOptimalPointMatching, 15, 21, 23, 24
generatePointDistributionStrategies, 16
getTSPInstancesOverview, 9, 16
getValidEdgeWeightTypes, 17
ggplot, 5, 23, 24

hasDepots, 17

importFromFile, 18
importFromTSPLibFormat, 18

isEuclidean, 19

isNetwork, 19

makeNetwork, 20
maximinLHS, 10
morphInstances, 21, 24

quoted(.), 2

rescaleNetwork, 22

visualizeMorphing, 21, 23, 24
visualizePointMatching, 15, 21, 24