Package ‘netgen’
February 20, 2015

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 1.0
Date 2015-01-20
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)
Imports ggplot2, lpSolve, checkmate (>= 1.5.1)
Suggests testthat
Author Jakob Bossek [aut, cre]
NeedsCompilation no
Repository CRAN
Date/Publication 2015-02-03 23:18:48

R topics documented:

as.character.Network .................................................. 2
as.data.frame.Network .................................................. 3
as.matrix.Network ..................................................... 3
autoplot.Network ....................................................... 4
dynamise ................................................................. 5
generateClusteredNetwork ............................................. 6
generateGridNetwork .................................................. 7
generateRandomNetwork .............................................. 8
getDepotCoordinates .................................................... 9
as.character.Network

Get basic network information as a string.

Description

Get basic network information as a string.

Usage

```r
## 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
```r
## S3 method for class 'Network'
as.data.frame(x, row.names = NULL, optional = FALSE,
              includeextras = TRUE, ...)
```

Arguments
- **x**  
  [Network]  
  Cluster instance.
- **row.names**  
  [character]  
  Row names for the result. Default is NULL.
- **optional**  
  [any]  
  Currently not used.
- **includeextras**  
  [logical(1)]  
  Include additional information like 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.

as.matrix.Network  Convert (clustered) instance to matrix.

Description
Convert (clustered) instance 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.
**dynamise**

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`
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. 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", ...)
```

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. Most often you want to generate 2-dimensional instances in the euclidean plane. Thus `2L` is the default setting.
- `generator` [function] Function which generates cluster centers. Default is `maximinLHS`.
- `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 distance to the nearest neighbour cluster center as diagonal elements.

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)
```
**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 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”.

... [any]
Currently not used.

**Value**

ClusteredNetwork Object of type ClusteredNetwork.

**See Also**

- `generateRandomNetwork`

**Examples**

```r
x = generateClusteredNetwork(n.points = 20L, n.cluster = 2L)
y = generateClusteredNetwork(n.points = 40L, n.cluster = 3L, n.depots = 2L)
```

**generateGridNetwork**

Generates a grid network.

**Description**

Generates a grid network.
Usage

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

Arguments

- `n.points.per.dim` [integer(1)]
  Number of points in each dimension.
- `n.dim` [integer(1)]
  Number of dimensions. Default ist 2.
- `lower` [numeric(1)]
  Lower box constraint of cube. Default is 0.
- `upper` [numeric(1)]
  Upper box constraint of cube. Default is 100.

Value

Network

Note

Grid networks with depots are not supported at the moment.

Examples

```r
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

```r
generateRandomNetwork(n.points, n.dim = 2L, n.depots = NULL, lower = 0,
upper = 100)
```
getDepotCoordinates

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.

Value
Network

Examples

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

getDepotCoordinates Get coordinates of depots.

Description

Get coordinates of depots.

Usage

generateRandomNetwork(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**

Return number of nodes of a network.

**Description**

Return 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(coords1, coords2)`

**Arguments**

`coords1`  
[matrix]  
Matrix of coordinates of the first point set.

`coords2`  
[matrix]  
Matrix of coordinates of the second point set.

**Value**

matrix Each row consists of the indices 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

----

hasDepots

*Check if network has depots.*

**Description**

Check if network has depots.

**Usage**

hasDepots(x)

**Arguments**

x [Network] Network.

**Value**

logical(1)
**makeClusteredNetwork**  
Generates clustered network.

**Description**  
Generates clustered network.

**Usage**  
```r
makeClusteredNetwork(coordinates, depot.coordinates = NULL, membership, lower = 0, upper = 100)
```

**Arguments**
- `coordinates`  
  Numeric matrix of 2D coordinates.
- `depot.coordinates`  
  Numeric matrix of 2D coordinates of depots. Default is NULL, which means no depots at all.
- `membership`  
  Vector of memberships.
- `lower`  
  Lower box constraint of cube. Default is 0.
- `upper`  
  Upper box constraint of cube. Default is 100.

**Value**
ClusteredNetwork

---

**makeNetwork**  
Generate network based on coordinates.

**Description**  
Generate network based on coordinates.

**Usage**  
```r
makeNetwork(coordinates, depot.coordinates = NULL, lower = 0, upper = 100)
```
**Arguments**

coordinates [matrix] Numeric matrix of 2D coordinates.
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 an integer programming procedure. If both instances contain depots, point matching is done separately on depots and the remaining nodes.

**Usage**

morphInstances(x, y, alpha)

**Arguments**

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

**Value**

Network Morphed network
visualizeMorphing

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)

### 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  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]  Network.
- **y**  [Network]  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.

Value

ggplot

See Also

morphInstances

visualizePointMatching

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

as.character.Network, 2
as.data.frame.Network, 3
as.matrix.Network, 3
autoplot.Network, 4
dynamise, 5
facet_grid, 16
generateClusteredNetwork, 5, 6
generateGridNetwork, 7
generateRandomNetwork, 5, 7, 8
getDepotCoordinates, 9
getNumberOfClusters, 10
getNumberOfDepots, 10
getNumberOfNodes, 11
getOptimalPointMatching, 11, 15, 16
getPointDistributionStrategies, 12
ggplot, 4, 5, 16

hasDepots, 12

makeClusteredNetwork, 13
makeNetwork, 13
maximinLHS, 6
morphInstances, 14, 16

visualizeMorphing, 15, 15, 16
visualizePointMatching, 11, 15, 16