Package 'geoGraph'

July 1, 2010

Version 1.0-0

Date 2010/07/01

Title geoGraph: implementing geographic graphs for large-scale spatial modelling.

Author Thibaut Jombart, Francois Balloux and Andrea Manica

Maintainer Thibaut Jombart <tjombart@imperial.ac.uk>

Suggests

Depends methods, graph, RBGL, sp, maptools, MASS, fields

Description .

License GPL (>=2)

LazyLoad yes

Collate classes.R basicMethods.R accessors.R auxil.R connectivity.R dropDead.R closestNode.R isInArea.R rebuild.R plot.R zoom.R findLand.R extractFromLayer.R interact.R setCosts.R dijkstra.R setDistCosts.R zzz.R

R topics documented:

`geoGraph-package` *The geoGraph package*

Description

This package implements classes and methods for large-scale georeferenced data handled through spatial graphs.

Main functionalities of `geoGraph` are summarized below.

=== DATA HANDLING ===
In `geoGraph`, data are stored as a particular formal class named gGraph. This class contains spatial coordinates of a set of nodes (@coords), attributes for these nodes (@nodes.attr), meta-information about nodes attributes (@meta), and a graph of connections between nodes of class graphNEL (@graph).

Several functions are available for handling gGraph data:

- some accessors allow to access slots of an object, sometimes with additional treatment of information: `getGraph`, `getNodesAttr`, `getCoords`, `getNodes`, `getEdges`, `getCosts`.

- `setEdges`: add/remove edges specified edges.

- `setCosts`: set costs of edges.

- `hasCosts`: tests if the graph is weighted (i.e., has non-uniform costs).

- `isInArea`: finds which nodes are in the currently plotted area.

- `areConnected`: tests if nodes are directly connected.

- `connectivityPlot`: plot connected components with different colors.

- `dropDeadEdges`: suppress edges whose weight is null.

- closestNode: given a longitude and a latitude, finds the closest node; specific values of node attribute can be provided, for instance, to find the closest node on land.

- show: printing of gGraph objects.

- extractFromLayer: extract information from GIS layers.

- findLand: checks which nodes are on land.

- setCosts: define edges weights accoring to rules specified in the @meta slot.

- geo.add.edges, geo.remove.edges: graphical functions for adding or removing edges.

- geo.change.attr: graphical functions for changing attributes of nodes.

=== GRAPHICS ===
geoGraph aims at providing advanced graphical facilities, such as zooming in or out particular area, moving the plotted area, or visualizing connectivity between nodes.

- plot: plot method with various options, allowing to display a shapefile (by default, the map of the world), using color according to attributes, showing connectivity between nodes, etc.

- points: similar to plot method, except that a new plot is not created.

- plotEdges: the specific function plotting edges. It detects if the object is a weighted graph, and plots edges accordingly.

- geo.zoomin, geo.zoomout: zoom in and out a plot.

- geo.back: replot the previous screens.

- geo.slide: slide the plotted area toward the indicated direction.

- geo.bookmark, geo.goto: set and goto a bookmarked area.

=== DATASETS ===
Datasets occupy a central place in geoGraph, since they provide the spatial models used in later operations.

Two main datasets are proposed, each being a gGraph resulting from the spliting of the earth into cells of (allmost perfectly) equal sizes. Two different resolutions are provided:
- worldgraph.10k: coverage using about 10,000 nodes
- worldgraph.40k: coverage using about 40,000 nodes

Other datasets are:
- worldshape: shapefile containing world countries.
- globalcoord.10k: spatial coordinates used in worldgraph.10k.

- globalcoord.40k: spatial coordinates used in worldgraph.40k.

To cite geoGraph, please use the reference given by `citation("geoGraph")`.

Details

Package:	geoGraph
Type:	Package
Version:	1.0-0
Date:	2010-07-01
License:	GPL (>=2)

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk> (maintainer)
François Balloux
Andrea Manica

Examples

```
## the class gGraph
data(worldgraph.10k)
worldgraph.10k

## plotting the object
plot(worldgraph.10k, reset=TRUE)

## zooming in
geo.zoomin(list(x=c(-6,38), y=c(35,73)))
title("Europe")

## to play interactively with graphics, use:
# geo.zoomin()
# geo.zoomout()
# geo.slide()
# geo.back()

## defining a new object restrained to visible nodes
x <- worldgraph.10k[isInArea(worldgraph.10k)]
plot(x,reset=TRUE, edges=TRUE)
title("x does just contain these visible nodes.")

## define weights for edges
x <- setCosts(x, attr.name="habitat", method="prod")
plot(x,edges=TRUE)
title("connectivity defined by habitat (land/land=1, other=0)")

## drop 'dead edges' (i.e. with weight 0)
x <- dropDeadEdges(x)
plot(x,edges=TRUE)
title("after droping edges with null weight")
```

`Auxiliary methods` *Auxiliary methods for geoGraph*

Description

These methods are low-level functions called by other procedures of `geoGraph`. Some can, however, be useful in themselves. Note that unlike other functions in `geoGraph`, these functions do not generally test for the validity of the provided arguments (for speed purposes).

- `hasCosts`: tests whether a gGraph has costs associated to its edges.

- `geo.segments`: a substitute to `segments` which correctly draws segments between locations distant by more than 90 degrees of longitude.

- `rebuild`: in development.

Usage

```
hasCosts(x)
geo.segments(x0, y0, x1, y1, col = par("fg"), lty = par("lty"),
    lwd = par("lwd"), ...)
```

Arguments

`x`	a valid gGraph.
`x0,y0`	coordinates of points *from* which to draw.
`x1,y1`	coordinates of points *to* which to draw.
`col`	a character string or an integer indicating the color of the segments.
`lty`	a character string or an integer indicating the type of line.
`lwd`	an integer indicating the line width.
`...`	further graphical parameters (from 'par') passed to the `segments` function.

Value

For `hasCost`, a logical value is returned. `geo.segments` returns NULL.

Author(s)

Thibaut Jombart (`<t.jombart@imperial.ac.uk>`)

Examples

```
data(worldgraph.10k)
hasCosts(worldgraph.10k)
```

`closestNode` *Find the closest node to a given location*

Description

The function `closestNode` searches for the closest node in a gGraph or a gData object to a given location. It is possible to restrain the research to given values of a node attribute. For instance, one can search the closest node on land to a given location.

This function is also used to match locations of a gData object with nodes of the `gGraph` object to which it is linked.

Usage

```
closestNode(x, ...)
## S4 method for signature 'gGraph':
closestNode(x, loc, zoneSize=5, attr.name=NULL, attr.values=NULL)
## S4 method for signature 'gData':
closestNode(x, zoneSize=5, attr.name=NULL, attr.values=NULL)
```

Arguments

`x`	a valid gGraph or gData object. In the latter case, the gGraph to which the gData is linked has to be in the current environment.
`...`	further arguments passed to specific methods.
`loc`	locations, specified as a list with two components indicating longitude and latitude of locations. Alternatively, this can be a data.frame or a matrix with longitude and latitude in columns, in this order. Note that `locator()` can be used to specify interactively the locations.
`zoneSize`	a numeric value indicating the size of the zone (in latitude/longitude units) where the closest node is searched for. Note that this only matters for speed purpose: if no closest node is found inside a given zone, the zone is expanded until nodes are found.
`attr.name`	the optional name of a node attribute. See details.
`attr.values`	an optional vector giving values for `attr.names`. See details.

Details

When creating a gData object, if the `gGraph.name` argument is provided, then locations are matched with the `gGraph` object automatically, by an internal call to closestNode. Note, however, that it is not possible to specify node attributes (`attr.names` and `attr.values`) this way.

Value

If `x` is a gGraph object: a vector of node names.

If `x` is a gData object: a gData object with matching nodes stored in the `@nodes.id` slot. Note that previous content of `@nodes.id` will be erased.

Author(s)

Thibaut Jombart (<t.jombart@imperial.ac.uk>)

See Also

geo.add.edges and geo.remove.edges to interactively add or remove edges in a gGraph object.

Examples

```
## Not run:
## interactive example ##
data(worldgraph.10k)
plot(worldgraph.10k, reset=TRUE)

## zooming in
geo.zoomin(list(x=c(-6,38), y=c(35,73)))
title("Europe")

## click some locations
myNodes <- closestNode(worldgraph.10k,locator(), attr.name="habitat", attr.value="land")
myNodes

## here are the closestNodes
points(getCoords(worldgraph.10k)[myNodes,], col="red")

## End(Not run)

## example with a gData object ##
data(worldgraph.10k)

myLoc <- list(x=c(3, -8, 11, 28), y=c(50, 57, 71, 67)) # some locations
obj <- new("gData", coords=myLoc) # new gData object
obj

obj@gGraph.name <- "worldgraph.10k" # this could be done when creating obj
obj <- closestNode(obj, attr.name="habitat", attr.value="land")

## plot the result (original location -> assigned node)
plot(obj, method="both")
title("'x'=location, 'o'=assigned node")
```

---

connectivity-checks

Check connectivity of a gGraph object

Description

The functions areNeighbours, areConnected and the method isConnected test connectivity in different ways.

- `areNeighbours`: tests connectivity between couples of nodes on an object inheriting `graph` class (like a graphNEL object).

- `areConnected`: tests if a set of nodes form a connected set on a gGraph object.

- `isConnected`: tests if the nodes of a gData object form a connected set. Note that this is a method for gData, the generic being defined in the `graph` package.

- `isReachable`: tests if one location (actually, the closest node to it) is reachable from the set of nodes of a gData object.

- `connectivityPlot`: plots connected sets of a gGraph or a gData object with different colors.

Usage

```
areNeighbours(V1, V2, graph)
areConnected(x, nodes)
## S4 method for signature 'gData':
isConnected(object, ...)
isReachable(x, loc)
## S4 method for signature 'gGraph':
connectivityPlot(x, ..., seed=NULL)
## S4 method for signature 'gData':
connectivityPlot(x, col.gGraph=0, ..., seed=NULL)
```

Arguments

`V1`	a vector of node names
`V2`	a vector of node names
`graph`	a valid graphNEL object.
`x`	a valid gGraph object.
`nodes`	a vector of node names
`object`	a valid gData object.
`...`	other arguments passed to other methods.
`loc`	location, specified as a list of two components giving respectively the longitude and the latitude. Alternatively, it can be a matrix-like object with one row and two columns.
`seed`	an optional integer giving the seed to be used when randomizing colors. One given seed will always give the same set of colors. NULL by default, meaning colors are randomized each time a plot is drawn.
`col.gGraph`	a character string or a number indicating the color of the nodes to be used when plotting the gGraph object. Defaults to '0', meaning that nodes are invisible.

Details

In `connectivityPlot`, isolated nodes (i.e. belonging to no connected set of size > 1) are plotted in light grey.

Value

- areNeighbours: a vector of logical, having one value for each couple of nodes.

- areConnected: a single logical value, being TRUE if nodes form a connected set.

- isConnected: a single logical value, being TRUE if nodes of the object form a connected set.

Author(s)

Thibaut Jombart (<t.jombart@imperial.ac.uk>)

Examples

```
data(rawgraph.10k)
data(worldgraph.10k)
connectivityPlot(rawgraph.10k)
connectivityPlot(worldgraph.10k)
```

---

dijkstra-methods *Shortest path using Dijkstra algorithm*

Description

The methods dijkstraFrom and dijkstraBetween are wrappers of procedures implemented in RBGL package, designed for gGraph and gData object.

dijkstraFrom finds minimum costs paths to nodes from a given 'source' node.

dijkstraBetween finds minimum costs paths between all possible pairs of nodes given two sets of nodes.

All these functions return objects with S3 class "gPath". These objects can be plotted using plot.gPath.

as.dist.gPath extracts the pairwise distances from the gPath returned by dijkstraBetween and returns a dist object. Note that if the gPath does not contain pairwise information, a warning will be issued, but the resulting output will likely be meaningless.

Usage

```
  ## S4 method for signature 'gGraph':
dijkstraFrom(x, start, weights="default")
  ## S4 method for signature 'gData':
dijkstraFrom(x, start, weights="default")
  ## S4 method for signature 'gGraph':
dijkstraBetween(x, from, to)
  ## S4 method for signature 'gData':
dijkstraBetween(m, diag = FALSE, upper = FALSE)
  ## S3 method for class 'gPath':
```

```
plot(x, col="rainbow", lwd=3, ...)
  ## S3 method for class 'gPath':
as.dist(m, diag=FALSE, upper=FALSE, res.type=c("dist","vector"))
```

Arguments

x	a gGraph or a gData object. For plotting method of gPath objects, a gPath object.
start	a character string naming the 'source' node.
weights	an optional set of weights for the edges.
from	a vector of character strings giving node names.
to	a vector of character strings giving node names.
col	a character string indicating a color or a palette of colors to be used for plotting edges.
lwd	a numeric value indicating the width of edges.
m	a gPath object obtained by dijkstraBetween.
diag,upper	unused parameters added for consistency with as.dist.
res.type	a character string indicating what type of result should be returned: a dist object ('dist'), or a vector of distances ('vector'). Note that 'dist' should only be required for pairwise data, as output by dijkstraBetween (as opposed to dijkstraFrom).
...	further arguments passed to the segments method.

Details

In 'dijkstraBetween', paths are seeked all possible pairs of nodes between 'from' and 'to'.

Value

A "gPath" object. These are basically the outputs of RBGL's sp.between function (see ?sp.between), with a class attribute set to "gPath", and an additional slot 'xy' containing geographic coordinates of the nodes involved in the paths.

Author(s)

Thibaut Jombart (<t.jombart@imperial.ac.uk>)

Examples

```
## Not run:
data(worldgraph.40k)
data(hgdp)

## plotting
world <- worldgraph.40k
par(mar=rep(.1,4))
plot(world, reset=TRUE)

## check connectivity
isConnected(hgdp) # must be ok
```

```
## Lowest cost path from an hypothetical origin
ori.coord <- list(33,10) # one given location long/lat
points(data.frame(ori.coord), pch="x", col="black", cex=3) # an 'x' shows the putative or
ori <- closestNode(world, ori.coord) # assign it the closest node

myPath <- dijkstraFrom(hgdp, ori) # compute shortest path

## plotting
plot(world,pch="") # plot the world
points(hgdp, lwd=3) # plot populations
points(data.frame(ori.coord), pch="x", col="black", cex=3) # add origin
plot(myPath) # plot the path

## End(Not run)
```

`dropDeadEdges` *Get rid of some 'dead' edges or nodes*

Description

The functions `dropDeadEdges` and `dropDeadNodes` are used to remove 'dead edges' and 'dead nodes'.

Dead edges are edges associated to a prohibitive cost, that is, edges that no longer imply connectivity between two nodes.

Dead nodes are nodes that are not connected to any other node, thus not having any role in the connectivity of a graph.

Usage

```
dropDeadEdges(x, thres)
dropDeadNodes(x)
```

Arguments

`x`	a valid gGraph.
`thres`	a numeric value indicating the threshold cost for an edge to be removed. All costs strictly greater than `thres` will be removed.

Value

A gGraph object.

Author(s)

Thibaut Jombart (`<t.jombart@imperial.ac.uk>`)

Examples

```
## Not run:
data(worldgraph.10k)
plot(worldgraph.10k,reset=TRUE)
x <- dropDeadNodes(worldgraph.10k)
plot(x)

## End(Not run)
```

`extractFromLayer` *Retrieves node attributes from a layer*

Description

The generic function `extractFromLayer` uses information from a GIS shapefile to define node attributes. For each node, information is retrieved from the layer and assigned to that node.

Nodes can be specified in different ways, including by providing a gGraph or a gData object. Outputs match the input formats.

Usage

```
extractFromLayer(x, ...)
## S4 method for signature 'matrix':
extractFromLayer(x, layer="world", attr="all",...)
## S4 method for signature 'data.frame':
extractFromLayer(x, layer="world", attr="all",...)
## S4 method for signature 'list':
extractFromLayer(x, layer="world", attr="all",...)
## S4 method for signature 'gGraph':
extractFromLayer(x, layer="world", attr="all",...)
## S4 method for signature 'gData':
extractFromLayer(x, layer="world", attr="all",...)
```

Arguments

`x`	a matrix, a data.frame, a list, a valid gGraph, or a valid gData object. For matrix and data.frame, input must have two columns giving longitudes and latitudes of locations being considered. For list, input must have two components being vectors giving longitudes and latitudes of locations.
`layer`	a shapefile of the class `SpatialPolygonsDataFrame` (see `readShapePoly` in maptools package to import such data from a GIS shapefile). Alternatively, a character string indicating one shapefile released with geoGraph; currently, only 'world' is available (see `?data(worldshape)`).
`attr`	a character vector giving names of the variables to be extracted from the layer. If 'all', all available variables are extracted. In case of problem, available names are displayed with the error message. Available data are also stored in `layer@data`.
`...`	further arguments to be passed to other methds. Currently not used.

Value

The output depends on the nature of the input:
- `matrix, data.frame, list`: a data.frame with one row per location, and as many columns as requested variables ('attributes').

- `gGraph`: a gGraph object with new node attributes (`@nodes.attr` slot). If nodes attributes already existed, new attributes are added as new columns.

- `gData`: a gData object with new data associated to locations (`@data` slot). New information is merge to older information according to the type of data being stored.

Author(s)

Thibaut Jombart (<t.jombart@imperial.ac.uk>)

See Also

`findLand`, to find which locations are on land.

Examples

```
## Not run:
data(worldgraph.10k)
data(worldshape)

plot(worldgraph.10k, reset=TRUE)


## see what info is available
names(worldshape@data)
unique(worldshape@data$CONTINENT)


## retrieve continent info for all nodes
## (might take a few seconds)
x <- extractFromLayer(worldgraph.10k, layer=worldshape, attr="CONTINENT")
x
table(getNodesAttr(x, attr.name="CONTINENT"))


## subset Africa
temp <- getNodesAttr(x, attr.name="CONTINENT")=="Africa"
temp[is.na(temp)] <- FALSE
x <- x[temp]
plot(x, reset=TRUE)


## End(Not run)
```

`findInLayer` *Retrieves node attributes from a layer*

Description

The generic function `findInLayer` uses information from a GIS shapefile to define node attributes. For each node, information is retrieved from the layer and assigned to that node.

Nodes can be specified in different ways, including by providing a gGraph or a gData object. Outputs match the input formats.

Usage

```
findInLayer(x, ...)
## S4 method for signature 'matrix':
findInLayer(x, layer="world", attr="all",...)
## S4 method for signature 'data.frame':
findInLayer(x, layer="world", attr="all",...)
## S4 method for signature 'list':
findInLayer(x, layer="world", attr="all",...)
## S4 method for signature 'gGraph':
findInLayer(x, layer="world", attr="all",...)
## S4 method for signature 'gData':
findInLayer(x, layer="world", attr="all",...)
```

Arguments

`x`	a matrix, a data.frame, a list, a valid gGraph, or a valid gData object. For matrix and data.frame, input must have two columns giving longitudes and latitudes of locations being considered. For list, input must have two components being vectors giving longitudes and latitudes of locations.
`layer`	a shapefile of the class `SpatialPolygonsDataFrame` (see `readShapePoly` in maptools package to import such data from a GIS shapefile). Alternatively, a character string indicating one shapefile released with geoGraph; currently, only 'world' is available (see `?data(worldshape)`).
`attr`	a character vector giving names of the variables to be extracted from the layer. If 'all', all available variables are extracted. In case of problem, available names are displayed with the error message. Available data are also stored in `layer@data`.
`...`	further arguments to be passed to other methds. Currently not used.

Value

The output depends on the nature of the input:
- `matrix, data.frame, list`: a data.frame with one row per location, and as many columns as requested variables ('attributes').

- `gGraph`: a gGraph object with new node attributes (`@nodes.attr` slot). If nodes attributes already existed, new attributes are added as new columns.

- gData: a gData object with new data associated to locations (@data slot). New information is merge to older information according to the type of data being stored.

Author(s)

Thibaut Jombart (<t.jombart@imperial.ac.uk>)

See Also

findLand, to find which locations are on land.

Examples

```
## Not run:
data(worldgraph.10k)
data(worldshape)

plot(worldgraph.10k, reset=TRUE)


## see what info is available
names(worldshape@data)
unique(worldshape@data$CONTINENT)


## retrieve continent info for all nodes
## (might take a few seconds)
x <- findInLayer(worldgraph.10k, layer=worldshape, attr="CONTINENT")
x
table(getNodesAttr(x, attr.name="CONTINENT"))


## subset Africa
temp <- getNodesAttr(x, attr.name="CONTINENT")=="Africa"
temp[is.na(temp)] <- FALSE
x <- x[temp]
plot(x, reset=TRUE)


## End(Not run)
```

---

findLand *Find which nodes are on land*

Description

The generic function findLand uses information from a GIS shapefile to define which nodes are on land, and which are not. Strickly speaking, being 'on land' is in fact being inside a polygon of the shapefile.

Nodes can be specified either as a matrix of geographic coordinates, or as a gGraph object.

Usage

```
findLand(x, ...)
## S4 method for signature 'matrix':
findLand(x, shape="world", ...)
## S4 method for signature 'data.frame':
findLand(x, shape="world", ...)
## S4 method for signature 'gGraph':
findLand(x, shape="world", attr.name="habitat", ...)
```

Arguments

x	a matrix, a data.frame, or a valid gGraph object. For matrix and data.frame, input must have two columns giving longitudes and latitudes of locations being considered.
shape	a shapefile of the class SpatialPolygonsDataFrame (see readShapePoly in maptools package to import such data from a GIS shapefile). Alternatively, a character string indicating one shapefile released with geoGraph; currently, only 'world' is available (see ?data(worldshape)).
...	further arguments to be passed to other methods. Currently not used.
attr.name	a character string giving the name of the node attribute in which the output is to be stored.

Value

The output depends on the nature of the input:
- matrix, data.frame: a factor with two levels being 'land' and 'sea'.

- gGraph: a gGraph object with a new node attribute, possibly added to previously existing node attributes (@nodes.attr slot).

Author(s)

Thibaut Jombart (<t.jombart@imperial.ac.uk>)

See Also

extractFromLayer, to retrieve any information from a GIS shapefile.

Examples

```
data(worldshape)

## create a new gGraph with random coordinates
myCoords <- data.frame(long=runif(1000,-180,180), lat=runif(1000,-90,90))
obj <- new("gGraph", coords=myCoords)
obj # note: no node attribute
plot(obj)

## find which points are on land
obj <- findLand(obj)
obj # note: new node attribute
```

```
## define rules for colors
temp <- data.frame(habitat=c("land","sea"), color=c("green","blue"))
temp
obj@meta$color <- temp

## plot object with new colors
plot(obj)
```

---

gData-class *Formal class "gData"*

Description

The class gData is a formal (S4) class storing georeferenced data, consisting in a set of locations (longitude and latitude) where one or several variables have been measured. These data are designed to be matched against a gGraph object, each location being assigned to the closest node of the gGraph object.

Note that for several operations on a gData object, the gGraph object to which it is linked will have to be present in the same environment.

Objects from the class gData

gData objects can be created by calls to new("gData", ...), where '...' can be the following arguments:

coords a matrix of spatial coordinates with two columns, being respectively longitude (from -180 to 180) and latitude. Positive numbers are intended as 'east' and 'north', respectively.

nodes.id a vector of character strings giving the name of the nodes (of the gGraph object) associated to the locations.

data any kind of data associated to the locations in coords. For matrix-like objects, rows should correspond to locations.

gGraph.name a character string the name of the gGraph object against which the object is matched.

gGraph.version a character string being a syntaxically valid POSIXct expression (see ?POSIXct), indicating the version (i.e., precise date) of the gGraph object against which the object is matched.

Note that none of these is mandatory: new("gData") would work, and create an empty gGraph object. Also note that a finer matching of locations against the nodes of a gGraph object can be achieved after creating the object, for instance using the closestNode method.

Slots

coords: Object of class "matrix" storing geographic coordinates (see above).

nodes.id: Object of class "character" storing names of the nodes of a gGraph object assigned to the locations in @coords.

data: Any object where observations match locations in @coords.

gGraph.name: the name of a gGraph object (see above).

gGraph.version: the version of a gGraph object (see above).

Methods

Here is a list of methods available for `gData` objects. See corresponding manpages for further documentation. Specific manpages exist for accessors with more than one argument. These are indicated by a '*' symbol next to the method's name.

[`signature(x = "gData")`: usual method to subset objects in R. First indice corresponds to locations (it can be a logical, a numeric, or a character); when a character is used, it has to be node names matching those in `@nodes.id`. Optionnaly, a second indice can be provided to subset columns of `data` (i.e., variables associated to locations) as well.

closestNode `signature(x = "gData")`: find the node closest to a given location.

dijkstraBetween `signature(x = "gData")`: least-cost path between two sets of nodes.

dijkstraFrom `signature(x = "gData")`: least-cost path between a set of nodes and a 'source node'.

extractFromLayer `signature(x = "gData")`: retrieve information from a shapefile and add it to `nodes.attr`.

getCoords `signature(x = "gData")`: get geographic coordinates of the object. The additional argument `original` is a logical indicating if original coordinates should be returned (TRUE, default), or if coordinates of the assigned nodes of the `gGraph` should be returned.

getData `signature(x = "gData")`: get the data of the object (i.e., content of the slot `@data`).

getNodes `signature(x = "gData")`: get the names of the nodes of the object.

initialize `signature(.Object = "gData")`: internal definition of the constructor.

isInArea `signature(x = "gData")`: check which nodes are in a given rectangular area.

points `signature(x = "gData")`: add points to an existing plot.

show `signature(object = "gData")`: prints the object to screen.

isConnected `signature(object = "gData")`: check if nodes are a connected set.

getGraph `signature(object = "gData")`: get the graphNEL object linked to a `gData`.

getNodesAttr* `signature(x = "gData")`: get the nodes attributes of the object.

Author(s)

Thibaut Jombart (<t.jombart@imperial.ac.uk>)

See Also

Related class:
- `gGraph`

Examples

```
## load data
data(worldgraph.40k)
data(hgdp)
hgdp

## plot data
plot(worldgraph.40k, pch="")
points(hgdp)
```

```
## subset and plot data
onlyNorth <- hgdp[hgdp@data$Latitude >0] # only northern populations

plot(worldgraph.40k, reset=TRUE)
abline(h=0) # equator
points(onlyNorth, pch.node=20, cex=2, col.node="purple")
```

`geo.add.edges` *Add and remove edges from a gGraph object*

Description

The functions `geo.add.edges` and `geo.remove.edges` allow one to add or remove edges interactively with a gGraph object. When adding edges, two approaches are possible:
- click vertices defining new edges (mode="points")
- select an area in which all edges from a reference graph are added (mode="area").

Usage

```
geo.add.edges(x, mode=c("points","area","all"), refObj="rawgraph.40k")
geo.remove.edges(x, mode=c("points","area"))
```

Arguments

`x`	a valid gGraph object.
`mode`	a character string indicating the mode for addition or removal of edges. 'points': user is expected to click vertices to indicate edges. 'area': user is expected to click two points defining a rectangular area within which all edges are selected. 'all': all edges from the reference graph are added to the current object.
`refObj`	a valid gGraph object, used as a reference when adding edges. When selecting an area inside which edges are added, all edges existing in this area in `refObj` are added to `x`. Alternatively, a character string can be provided, corresponding to one of the following datasets: 'rawgraph.10k', rawgraph.40k'.

Value

A gGraph object with newly added or removed edges.

Author(s)

Thibaut Jombart (<t.jombart@imperial.ac.uk>)

See Also

`setEdges` to non-interactively add or remove edges in a gGraph object.

Examples

```
## Not run:
data(worldgraph.10k)
plot(worldgraph.10k, reset=TRUE)

## zooming in
geo.zoomin(list(x=c(-6,38), y=c(35,73)))
title("Europe")

## remove edges
geo.remove.edges(worldgraph.10k) # points mode
geo.remove.edges(worldgraph.10k, mode="area") # area mode

## add edges
geo.add.edges(worldgraph.10k) # points mode
geo.add.edges(worldgraph.10k, mode="area") # area mode

## End(Not run)
```

---

geo.change.attr *Change values of a node attribute*

Description

The functions `geo.change.attr` changes values of a given node attribute for a set of selected nodes of a gGraph object.

Usage

```
geo.change.attr(x, mode=c("points","area"), attr.name, attr.value,
                only.name=NULL, only.value=NULL, newCol="black",
                restore.edges=TRUE, refObj = "rawgraph.40k")
```

Arguments

`x`	a valid gGraph object.
`mode`	a character string indicating whether selected nodes are clicked one by one ('points') or by defining a rectangular area ('area').
`attr.name`	the name of the node attribute to be modified.
`attr.value`	the new value of attribute assigned to selected nodes.
`only.name`	(optional) in area mode, the name of a node attribute to add an extra selection criterion. See details.
`only.value`	(optional) in area mode, and if `only.name` is specified, the values of `only.name` that can be selected. See details.
`newCol`	a character string giving the new color for the attribute value.
`restore.edges`	a logical indicating whether edges stemming from the modified nodes should be re-added to the graph, using `refObj` as a reference. This is useful when connectivity is to be redefined using `setCosts` for nodes that were previously disconnected.

refObj — a character string or a gGraph object, used as reference when re-adding edges. If a character string is provided, it must match one of the following dataset: 'rawgraph.10k', 'rawgraph.40k'.

Details

The argument only.name allows one to perform a more accurate selection of nodes whose attribute is changed, by specifying values (only.value) of an attribute (only.name) that can be selected. For instance, one may want to define new attributes for nodes of worldgraph.10k that are exclusively on land: this would be done by specifying only.name="habitat" and only.value="land".

Value

A gGraph object with modified node attributes.

Author(s)

Thibaut Jombart (<t.jombart@imperial.ac.uk>)

Examples

```
## Not run:
data(worldgraph.10k)
plot(worldgraph.10k, reset=TRUE)

## have to click here for an area
## all nodes are modified in the area
x <- geo.change.attr(worldgraph.10k, mode="area", attr.name="habitat", attr.value="fancy
habitat", newCol="pink") # modify selected area

plot(x,reset=TRUE) # modification in the whole selected area

## have to click here for an area
## only nodes on land are modified
x <- geo.change.attr(x, mode="area", attr.name="habitat", attr.value="fancy2
habitat", newCol="purple", only.name="habitat", only.value="land")

plot(x,reset=TRUE) # modification in the whole selected area

## End(Not run)
```

---

geo.zoomin *Navigate in the plot of a gGraph object*

Description

The functions geo.zoomin, geo.zoomout, geo.slide, geo.back, geo.bookmark and geo.goto are used to navigate interactively in the plot of a gGraph object.

geo.zoomin and geo.zoomout are used to zoom in and out. For zooming in, the user has to delimit the opposite corner of the new plotting area; alternatively, a set of coordinates can be provided. For zooming out, each click on the screen will zoom out further.

`geo.slide` moves the window toward the direction indicated by clicking in the screen.

`geo.back` redraws previous plots each time screen is clicked.

`geo.bookmark` sets a bookmark for the current area. If the name for the bookmark is left to NULL, then the list of currently available bookmarks is returned.

`geo.goto` allows the user to get back to a bookmarked area.

`.zoomlog.up` is an auxiliary function used to update the zoom log, by providing new sets of coordinates.

Whenever clicking is needed, a right-click will stop the function.

Usage

```
geo.zoomin(reg=NULL)
geo.zoomout()
geo.slide()
geo.back()
geo.bookmark(name=NULL)
geo.goto(name)
.zoomlog.up(vec)
```

Arguments

`reg`	a list of length 2, with its first component being the new x (longitude) boundaries (a vector of length 2), and its second being new y (latitude) boundaries (a vector of length 2).
`vec`	a numeric vector of length 4 giving the new coordinates of the plotting window, in the order: xmin, xmax, ymin, ymax.
`name`	a character string giving the name of the bookmark to create (in `geo.bookmark`) or to get back to (in `geo.goto`).

Author(s)

Thibaut Jombart (<t.jombart@imperial.ac.uk>)

See Also

`plot.gGraph` for plotting of a gGraph object.

Examples

```
data(worldgraph.10k)
plot(worldgraph.10k, reset=TRUE)

## zooming in
x.ini <- c(-100,-60)
y.ini <- c(-30,30)
for(i in 0:3){
geo.zoomin(list(x=x.ini + i*60, y=y.ini))
}
```

```
## Not run:
## going back
geo.back() # you have to click !

## zooming in interactively
geo.zoomin() # you have to click !

## zooming out
geo.zoomout() # you have to click !

## moving window
geo.slide() # you have to click !

## End(Not run)
```

`getColors` *Get colors associated to edges of a gGraph object*

Description

The function `getColors` returns the colors associated to the nodes of a gGraph object, based on a specified node attribute.

Usage

```
getColors(x, ...)
## S4 method for signature 'gGraph':
getColors(x, nodes="all", attr.name, ...)
```

Arguments

`x`	a valid gGraph.
`nodes`	a vector of character strings or of integers identifying nodes by their name or their index. Can be "all", in which case all nodes are considered.
`attr.name`	a character string indicating the name of node attribute to be used to define colors.
`...`	other arguments passed to other methods.

Details

Colors are based on a node attribute, that is, on a column of the `nodes.attr` data.frame. This attribute should have a finite number of values, and would most likely be a factor. Correspondence between values of this variable and colors must be provided in the `@meta$color` slot. See example section to know how this slot should be designed.

Value

A vector of characters being valid colors.

Author(s)

Thibaut Jombart (<t.jombart@imperial.ac.uk>)

Examples

```
##
data(worldgraph.10k)

worldgraph.10k # there is a node attribute 'habitat'
worldgraph.10k@meta$color

head(getNodes(worldgraph.10k))
head(getColors(worldgraph.10k,res.type="vector", attr.name="habitat"))
```

---

getCosts *Get costs associated to edges of a gGraph object*

Description

The function getCosts returns the costs associated to the edges of a gGraph object using different possible outputs. These outputs are designed to match possible outputs of getEdges function.

Usage

```
getCosts(x, ...)
## S4 method for signature 'gGraph':
getCosts(x, res.type=c("asIs","vector"), unique=FALSE, ...)
```

Arguments

x	a valid gGraph.
res.type	a character string indicating which kind of output should be used. See value.
unique	a logical indicating whether the costs should be returned for unique edges (TRUE), or if duplicate edges should be considered as well (TRUE, default).
...	other arguments passed to other methods (currently unused).

Details

The notion of 'costs' in the context of gGraph objects is identical to the concept of 'weights' in graph (and thus graphNEL) objects. The larger it is for an edge, the less connectivity there is between the couple of concerned nodes.

Value

The output depends on the value of the argument res.type:
- asIs: output is a named list of weights, each slot containing weights associated to the edges stemming from one given node. This format is that of the weights accessor for graphNEL objects.

- vector: a vector of weights; this output matches matrix outputs of getEdges.

Author(s)

Thibaut Jombart (<t.jombart@imperial.ac.uk>)

See Also

Most other accessors are documented in gGraph manpage.

Examples

```
data(worldgraph.10k)

head(getEdges(worldgraph.10k, res.type="matNames",unique=TRUE))
head(getCosts(worldgraph.10k,res.type="vector",unique=TRUE))
```

---

getEdges *Get edges from a gGraph object*

Description

The function getEdges returns the edges of a gGraph object using different possible outputs.

Usage

```
getEdges(x, ...)
## S4 method for signature 'gGraph':
getEdges(x, res.type=c("asIs","matNames", "matId"), unique=FALSE, ...)
```

Arguments

x	a valid gGraph.
res.type	a character string indicating which kind of output should be used. See value.
unique	a logical indicating whether all returned edges should be unique (TRUE) or if duplicated edges should be allowed (TRUE, default).
...	other arguments passed to other methods (currently unused).

Value

The output depends on the value of the argument res.type:
- asIs: output is a named list of nodes, each slot containing nodes forming an edge with one given node. This format is that of the edges accessor for graphNEL objects.

- matNames: a matrix with two columns giving couples of node names forming edges.

- matId: a matrix with two columns giving couples of node indices forming edges.

Author(s)

Thibaut Jombart (<t.jombart@imperial.ac.uk>)

See Also

Most other accessors are documented in gGraph manpage.

See `setEdges` to add/remove edges, or `geo.add.edges` and `geo.remove.edges` for interactive versions.

Examples

```
example(gGraph)

getEdges(x)
getEdges(x,res.type="matNames")
getEdges(x,res.type="matId")
```

`getNodesAttr` *Get nodes attributes from gGraph/gData object*

Description

The function `getNodesAttr` returns the values of a set of variables associated to the nodes (i.e. node attributes) of a gGraph or gData object.

Usage

```
getNodesAttr(x, ...)
## S4 method for signature 'gGraph':
getNodesAttr(x, nodes=NULL, attr.name=NULL, ...)
## S4 method for signature 'gData':
getNodesAttr(x, attr.name=NULL, ...)
```

Arguments

`x`	a valid gGraph or gData object.
`nodes`	an optional integer, logical, or character string indicating the subset of nodes to be used. If NULL, all nodes are used.
`attr.name`	an optional character string indicating which node attributes should be returned. If provided, it must match at least one of the columns of `x@nodes.attr`.
`...`	other arguments passed to other methods (currently unused).

Value

A data.frame with the requested nodes attributes. Nodes are displayed in rows, variables in columns.

Author(s)

Thibaut Jombart (`<t.jombart@imperial.ac.uk>`)

See Also

Most other accessors are documented in gGraph and gData manpages.

Examples

```
## gGraph method
data(worldgraph.40k)
head(getNodesAttr(worldgraph.40k))


## gData method
data(hgdp)
getNodesAttr(hgdp)
```

`gGraph-class` *Formal class "gGraph"*

Description

The class `gGraph` is a formal (S4) class storing geographic data.
Such data are composed of a set of geographic coordinates of vertices (or 'nodes'), and a graph describing connectivity between these vertices. Data associated to the nodes can also be stored ('nodes attributes'), as well as meta-information used when plotting the object, or when computing weights associated to the edges based on nodes attributes.

In all slots, nodes are uniquely identified by their name (reference is taken from the row names of `@coords` slot).

Objects from the class gGraph

`gGraph` objects can be created by calls to `new("gGraph", ...)`, where '...' can be the following arguments:

`coords` a matrix of spatial coordinates with two columns, being respectively longitude (from -180 to 180) and latitude. Positive numbers are intended as 'east' and 'north', respectively.

`nodes.attr` a data.frame whose rows are nodes, and whose columns are different variables associated to the nodes.

`meta` a list, most likely containing named data.frames (see Slots).

`graph` an object of the class graphNEL, from the `graph` package (see `class?graphNEL`), describing connectivity among nodes.

Note that none of these is mandatory: `new("gGraph")` would work, and create an empty `gGraph` object.

Slots

`coords`**:** Object of class `"matrix"` storing geographic coordinates (see above).

`nodes.attr`**:** A `"data.frame"` storing nodes data (see above).

`meta`**:** A `"list"` containing meta information about how data in the `nodes.attr` slot should be used in different contexts. This information is to be stored as named data.frames. Currently recognized meta-information are:
- `$color`: a data.frame giving colors to be used when plotting the object, according to a node attribute taking a finite number of values (likely a factor). Each documented attribute is

a column, the last one being reserved to (valid) named colors. See the `@meta$color` slot in `worldgraph.10k` for an example.
- `$weights`: a data.frame giving weights to be used when computing edge weights using `setCosts`. The first column is a node attribute, and the second column gives corresponding weights. See the `@meta$costs` slot in `worldgraph.10k` for an example.

`graph`: Object of class graphNEL (see `class?graphNEL` for more information).

Methods

Here is a list of methods available for `gGraph` objects. Specific manpages exist for accessors with more than one argument. These are indicated by a '*' symbol next to the method's name.

[`signature(x = "gGraph")`: usual method to subset objects in R. First indice corresponds to nodes (it can be a logical, a numeric, or a character); optionnaly, a second indice can be provided to subset columns of `nodes.attr` (i.e., variables associated to nodes) as well.

closestNode `signature(x = "gGraph")`: find the node closest to a given location.

dijkstraBetween `signature(x = "gGraph")`: least-cost path between two sets of nodes.

dijkstraFrom `signature(x = "gGraph")`: least-cost path between a set of nodes and a 'source node'.

dropCosts `signature(x = "gGraph")`: remove costs associated to edges.

extractFromLayer `signature(x = "gGraph")`: retrieve information from a shapefile and add it to `nodes.attr`.

findLand `signature(x = "gGraph")`: find which nodes are on land, and which are not.

getColors* `signature(x = "gGraph")`: get colors corresponding to the nodes, as specified in the `@meta$color` slot.

getCoords `signature(x = "gGraph")`: get geographic coordinates of the object.

getDates `signature(x = "gGraph")`: get dates of modifications of the object.

getEdges* `signature(x = "gGraph")`: get edges of the object.

getGraph `signature(x = "gGraph")`: get the graph of the object (as a `graphNEL` object).

getNodesAttr* `signature(x = "gGraph")`: get the nodes attributes of the object.

getNodes `signature(x = "gGraph")`: get the names of the nodes of the object.

getCosts* `signature(x = "gGraph")`: get the weights of the edges of the object.

initialize `signature(.Object = "gGraph")`: internal definition of the constructor.

isInArea `signature(x = "gGraph")`: check which nodes are in a given rectangular area.

plot `signature(x = "gGraph", y = "missing")`: plot the object.

points `signature(x = "gGraph")`: add points to an existing plot.

setEdges* `signature(x = "gGraph")`: add/remove edges to/from the object.

show `signature(object = "gGraph")`: prints the object to screen.

Note that other functions not directly related to gGraph objects are of important use in `geoGraph`. See package documentation (`?geoGraph`) for a more complete list of implemented functions.

Author(s)

Thibaut Jombart (`<t.jombart@imperial.ac.uk>`)

See Also

Related classes are:

- graphNEL (graph package): slot @graph in gGraph.

Examples

```
## create an empty object
new("gGraph", comments="This is my first gGraph")


## plotting the object
data(rawgraph.10k)
plot(rawgraph.10k, reset=TRUE)

## zooming in
geo.zoomin(list(x=c(-6,38), y=c(35,73)))
title("Europe")

## to play interactively with graphics, use:
# geo.zoomin()
# geo.zoomout()
# geo.slide()
# geo.back()

## defining a new object restrained to visible nodes
x <- rawgraph.10k[isInArea(rawgraph.10k)]
plot(x,reset=TRUE, edges=TRUE)
title("x does just contain these visible nodes.")

## define weights for edges
x <- setCosts(x, attr.name="habitat", method="prod")
plot(x,edges=TRUE)
title("costs defined by habitat (land/land=1, other=100)")

## drop 'dead edges' (i.e. with weight 0)
x <- dropDeadEdges(x, thres=10)
plot(x,edges=TRUE)
title("after droping edges with null weight")
```

---

gGraphHistory *Disabled functionality*

Description

Disabled functionality

hgdp *Human genome diversity panel - georeferenced data*

Description

The datasets `hgdp` and `hgdpPlus` provides genetic diversity several human populations worldwide. Both datasets are gData objects, interfaced with the gGraph object `worldgraph.40k`.

`hgdp` describes 52 populations from the original Human Genome Diversity Panel.

`hgdpPlus` describes `hgdp` populations plus 24 native American populations.

Usage

```
data(hgdp)
data(hgdpPlus)
```

Format

`hgdp` is a gGraph object with the following data:

References

Authors *Journal*, YEAR, **nb**: pp-pp.

Examples

```
data(hgdp)
data(worldgraph.40k)
hgdp

## plotting the object
plot(hgdp)


## results from Handley et al.
## Not run:
## Addis Ababa
addis <- list(lon=38.74,lat=9.03)
addis <- closestNode(worldgraph.40k,addis) # this takes a while

## shortest path from Addis Ababa
myPath <- dijkstraFrom(hgdp, addis)

## plot results
plot(worldgraph.40k, col=0)
points(hgdp)
points(worldgraph.40k[addis], psize=3,pch="x", col="black")
plot(myPath)

## correlations distance/genetic div.
```

```
geo.dist <- sapply(myPath[-length(myPath)],function(e) e$length)
gen.div <- getData(hgdp)[,"Genetic.Div"]
plot(gen.div~geo.dist)
lm1 <- lm(gen.div~geo.dist)
abline(lm1, col="blue") # this regression is wrong
summary(lm1)

## End(Not run)
```

`installDep.geoGraph` *Install dependencies for geoGraph*

Description

This simple function installs the latest versions of the packages `graph` and `RBGL` on Bioconductor. This function requires a working internet connection, as well as administrator rights for the directory where the libraries are installed.

Usage

```
installDep.geoGraph()
```

Author(s)

Thibaut Jombart (<t.jombart@imperial.ac.uk>)

`isInArea` *Find which nodes fall in a given area*

Description

The generic function `isInArea` finds which nodes fall in a given area. Nodes can be specified in different ways, including by providing a gGraph or a gData object. Different format for the output are also available.

Usage

```
isInArea(x, ...)
## S4 method for signature 'matrix':
isInArea(x, reg="current", res.type=c("logical","integer","character"), buffer=0
## S4 method for signature 'data.frame':
isInArea(x, reg="current", res.type=c("logical","integer","character"), buffer=0
## S4 method for signature 'gGraph':
isInArea(x, reg="current", res.type=c("logical","integer","character"), buffer=0
## S4 method for signature 'gData':
isInArea(x, reg="current", res.type=c("logical","integer","character"), buffer=0
```

Arguments

x	a matrix, a data.frame, a valid gGraph, or a valid gData object. For matrix and data.frame, input must have two columns giving longitudes and latitudes of locations being considered.
...	further arguments passed to specific methods.
reg	a character string or a list indicating the area ('reg' stands for 'region'). Character strings can be "current" (current user window, default) or "zoom" (current zoom). If the argument is a list, is has to have two components, both being numeric vectors of length two, giving x and y limits of the area. Note that such list can be produced by `locator`, so `locator(1)` is a valid value for `reg`.
res.type	a character string indicating what kind of output should be produced. See value.
buffer	a numeric value giving a buffer adding extra space aroung the area, as a proportion of current area's dimensions.

Value

The output depends on the value of the argument `res.type`:
- `logical`: a vector of logicals having one value for each node of the input.

- `integer`: a vector of integers corresponding to the indices of nodes falling within the area.

- `character`: a vector of characters corresponding to the names of the nodes falling within the area.

Author(s)

Thibaut Jombart (<t.jombart@imperial.ac.uk>)

Examples

```
data(worldgraph.10k)
plot(worldgraph.10k, reset=TRUE)

## zooming in
geo.zoomin(list(x=c(-6,38), y=c(35,73)))
title("Europe")


## different outputs of isInArea
head(isInArea(worldgraph.10k)) # logical
length(isInArea(worldgraph.10k))
sum(isInArea(worldgraph.10k))
head(which(isInArea(worldgraph.10k))) # which nodes are TRUE ?

head(isInArea(worldgraph.10k, res.type="integer")) # node indices

head(isInArea(worldgraph.10k, res.type="character")) # node names


## use isInArea to have a subset of visible nodes
x <- worldgraph.10k[isInArea(worldgraph.10k)]
plot(x, reset=TRUE)
```

plot-gData *Plot a gData object.*

Description

Various functions to plot a gData object: `plot` opens a device and plots the object, while `points` plots the object on the existing device. Plotting of gData object relies on plotting the gGraph object to which it is linked, and then represent the locations of the gData and/or the associated nodes.

Usage

```
 ## S4 method for signature 'gData':
plot(x, type=c("nodes","original","both"),
                        pch.ori=4, pch.nodes=1,
                        col.ori="black", col.nodes="red",
                        col.gGraph=NULL, reset=FALSE,
                        sticky.points=TRUE,...)

 ## S4 method for signature 'gData':
points(x, type=c("nodes","original","both"),
                          pch.ori=4, pch.nodes=1,
                          col.ori="black", col.nodes="red",
                          sticky.points=TRUE,...)
```

Arguments

`x`	a valid gData object. The gData object to which it is linked must exist in the global environment.
`type`	a character string indicating which information should be plotted: original locations ('original'), associated nodes ('nodes', default), or both ('both'). In the latter case, an arrow goes from locations to nodes.
`pch.ori`	a numeric or a character indicating the type of point for locations.
`pch.nodes`	a numeric or a character indicating the type of point for nodes.
`col.ori`	a character string indicating the color to be used for locations.
`col.nodes`	a character string indicating the color to be used for nodes.
`col.gGraph`	a (recycled) color vector for the associated gGraph object. If NULL, default color is used. Set to `NA` or "transparent" to avoid plotting the gGraph.
`reset`	a logical stating whether the plotting area should be reset to fit the `gData` object (TRUE), or should conserve previous plotting and settings (FALSE, default).
`sticky.points`	a logical indicating if added points should be kept when replotting (TRUE, default), or not (FALSE). In any case, `reset=TRUE` will prevent points to be redrawn.
`...`	further arguments passed to `points`.

Details

When `sticky.points` is set to TRUE, all operations performed on the graphics like zooming or sliding the window can be performed without loosing the `gData` plot.

Author(s)

Thibaut Jombart (<t.jombart@imperial.ac.uk>)

See Also

- Different functions to explore these plots:
geo.zoomin, geo.zoomout, geo.slide, geo.back, geo.bookmark, geo.goto.

Examples

```
data(worldgraph.10k)

myLoc <- list(x=c(3, -8, 11, 28), y=c(50, 57, 71, 67)) # some locations
obj <- new("gData", coords=myLoc) # new gData object
obj

obj@gGraph.name <- "worldgraph.10k"
obj <- closestNode(obj, attr.name="habitat", attr.value="land")

## plot the result (original location -> assigned node)
plot(obj, type="both",reset=TRUE)
title("'x'=location, 'o'=assigned node")

## using different parameters
points(obj, type="both", pch.ori=2, col.ori="red", pch.nodes=20, col.nodes="pink")

## only nodes, fancy plot
plot(obj, col.nodes="red", cex=1, pch.node=20)
points(obj, col.nodes="red", cex=2)
points(obj, col.nodes="orange", cex=3)
points(obj, col.nodes="yellow", cex=4)
```

---

plot-gGraph *Plot a gGraph object.*

Description

Various functions to plot a gGraph object: plot opens a device and plot the object, while points plots the object on the existing device. plotEdges only plots the edges of the graph: it can be called directly, or via arguments passed to plot and points.

Plotting of a gGraph object stores some parameters in R; see details for more information.

Usage

```
  ## S4 method for signature 'gGraph':
plot(x, shape="world", psize=NULL, pch=19,
  col=NULL, edges=FALSE, reset=FALSE, bg.col="gray", border.col="dark
  gray", lwd=1, useCosts=NULL, maxLwd=3, col.rules=NULL, ...)
```

```
 ## S4 method for signature 'gGraph':
points(x, psize=NULL, pch=NULL, col=NULL,
 edges=FALSE, lwd=1, useCosts=NULL, maxLwd=3, col.rules=NULL,
 sticky.points=FALSE,...)

plotEdges(x, useCosts=NULL,
col="black", lwd=1, lty=1, pch=NULL, psize=NULL, pcol=NULL, maxLwd=3,
col.rules=NULL, sticky.edges=FALSE, ...)
```

Arguments

x	a gGraph object.
shape	a shapefile used as background to the object. Must be of the class SpatialPolygonsDataFrame (see readShapePoly in maptools package to import such data from a GIS shapefile). Alternatively, a character string indicating one shapefile released with geoGraph.
psize	a numeric giving the size of points.
pch	a numeric or a character indicating the type of point.
col	a character string indicating the color to be used.
edges	a logical indicating if edges should be plotted (TRUE) or not (FALSE).
reset	a logical indicating if plotting parameters should be reset (TRUE) or not (FALSE).
bg.col	a character string indicating the color of the polygons of the shapefile used as background.
border.col	a character string indicating the color of the polygon borders.
lwd	a numeric indicating the width of line (used for edges).
useCosts	a logical indicating if edge width should be inversely proportionnal to edge cost (TRUE) or not (FALSE).
maxLwd	a numeric indicating the maximum edge width (corresponding to the maximum weight).
col.rules	a data.frame with two named columns, the first one giving values of a node attribute, and the second one stating colors to be used for each value. If not provided, this is seeked from the @meta$color slot of the object.
sticky.points	a logical indicating if added points should be kept when replotting (TRUE), or not (FALSE). In any case, reset=TRUE will prevent points to be redrawn.
lty	the type of line (for the edges).
pcol	a character indicating the color to be used for points.
sticky.edges	a logical indicating whether added edges should be kept when replotting (TRUE), or not (FALSE, default). In any case, reset=TRUE will prevent points to be redrawn.
...	further arguments passed to the generic methods (plot, points, and segments, respectively).

Details

To be able to zoom in and out, or slide the window, previous plotting information are stored in a particular environment (.geoGraphEnv), which is created when loading `geoGraph`. Users should not have to interact directly with objects in this environment.

The resulting plotting behaviour is that when plotting a `gGraph` object, last plotting parameters are re-used. To override this behaviour, specify `reset=TRUE` as argument to `plot`.

Author(s)

Thibaut Jombart (<t.jombart@imperial.ac.uk>)

See Also

- Different functions to explore these plots:
`geo.zoomin`, `geo.zoomout`, `geo.slide`, `geo.back`.

- `isInArea`, to retain a set of visible data.

Examples

```
data(worldgraph.10k)

## just the background
plot(worldgraph.10k,reset=TRUE,type="n")

## basic plot
plot(worldgraph.10k)

## zooming and adding edges
geo.zoomin(list(x=c(90,150),y=c(0,-50)))
plot(worldgraph.10k, edges=TRUE)


## display edges differently
plotEdges(worldgraph.10k, col="red", lwd=2)


## replot points with different color
points(worldgraph.10k, col="orange")

## mask points in the sea
inSea <- unlist(getNodesAttr(worldgraph.10k,attr.name="habitat"))=="sea"
head(inSea)
points(worldgraph.10k[inSea], col="white", sticky=TRUE) # this will stay

## but better, only draw those on land, and use a fancy setup
par(bg="blue")
plot(worldgraph.10k[!inSea], bg.col="darkgreen", col="purple", edges=TRUE)
```

setCosts *Set friction in a gGraph object*

Description

The function setCosts define costs for the edges of a gGraph object according to a node attribute and some rules defined in the @meta$costs slot of the object. Each node has a value for the chosen attribute, which is associated to a costs (a friction). The cost of an edge is computed as a function (see argument method) of the costs of its nodes.

Note that costs are inversely proportionnal to connectivity between edges: the larger the cost associated to an edge, the lower the connectivity between the two concerned nodes.

Also note that 'costs' defined in geoGraph are equivalent to 'weights' as defined in graph and RBGL packages.

Usage

```
setCosts(x, attr.name=NULL, node.costs=NULL, method=c("mean", "product"))
```

Arguments

x	a gGraph object with a least one node attribute, and a @meta$costs component (for an example, see worldgraph.10k dataset).
attr.name	the name of the node attribute used to compute costs (i.e., of one column of @nodes.attr).
node.costs	a numeric vector giving costs associated to the nodes. If provided, it will be used instead of attr.name.
method	a character string indicating which method should be used to compute edge cost from nodes costs. Currently available options are 'mean' and 'prod', where the cost associated to an edge is respectively computed as the mean, or as the product of the costs of its nodes.

Value

A gGraph object with the newly defined costs used as weightings of edges.

Author(s)

Thibaut Jombart (<t.jombart@imperial.ac.uk>)

See Also

dropDeadEdges, to get rid of edge whose cost is below a given threshold. geo.add.edges to add edges to a gGraph object.

Examples

```
data(rawgraph.10k)
plot(rawgraph.10k, reset=TRUE)

## zooming in
geo.zoomin(list(x=c(-6,38), y=c(35,73)))
title("Europe")

## defining a new object restrained to visible nodes
x <- rawgraph.10k[isInArea(rawgraph.10k)]

## define weights for edges
x <- setCosts(x, attr.name="habitat")
plot(x,edges=TRUE)
title("costs defined by habitat (land/land=1, other=100)")
```

---

setDistCosts *Set costs associated to edges based on geographic distances*

Description

The function `setDistCosts` sets the costs of a gGraph object using the geographic distance. The cost associated to an edge is defined as the great circle distance between the two nodes of this edge. `setDistCosts` actually relies on `rdist.earth` of the `fields` package.

Usage

```
setDistCosts(x, ...)
## S4 method for signature 'gGraph':
setDistCosts(x, ...)
```

Arguments

`x`	a valid gGraph.
`...`	other arguments passed to other methods (currently unused).

Details

The notion of 'costs' in the context of gGraph objects is identical to the concept of 'weights' in graph (and thus graphNEL) objects. The larger it is for an edge, the less connectivity there is between the couple of concerned nodes.

Value

For the gGraph method, a gGraph object with appropriate weights. Note that former weights will be removed from the object.

Author(s)

Thibaut Jombart (<t.jombart@imperial.ac.uk>)

See Also

The `getCosts` accessor, returning costs of the edges of a gGraph object in different ways.

Examples

```
if(require(fields)){
## load data
data(rawgraph.10k)
plot(rawgraph.10k,reset=TRUE)
geo.zoomin(list(x=c(110,150),y=c(-10,-40)))
plotEdges(rawgraph.10k)

## compute costs
x <- rawgraph.10k[isInArea(rawgraph.10k)]
x <- setDistCosts(x)

## replot edges
plotEdges(x) # no big differences can be seen
head(getCosts(x))
}
```

---

setEdges *Add and remove edges from a gGraph object*

Description

The function `setEdges` allows one to add or remove edges in a gGraph by directly specifying the relevant nodes, as a list or a data.frame. This low-level function is called by `geo.add.edges` and `geo.remove.edges`.

Usage

```
setEdges(x, ...)
## S4 method for signature 'gGraph':
setEdges(x, add=NULL, remove=NULL, costs=NULL, ...)
```

Arguments

`x`	a valid gGraph object.
`add`	a list or a dataframe containing node names of edges to be added. The first element of the list (or column of the data.frame) gives starting nodes of edges; the second gives ending nodes. Hence, the nodes of the i-th edge are `add[[1]][i]` and `add[[2]][i]` if `add` is a list, and `add[i,]` if `add` is a data.frame.
`remove`	same as `add` argument, but edges are removed.
`costs`	a numeric vector providing costs of the edges to be added. `costs[i]` is the weight of the i-th edge.
`...`	other arguments passed to other methods (currently unused).

Value

A gGraph object with newly added or removed edges.

Author(s)

Thibaut Jombart (<t.jombart@imperial.ac.uk>)

See Also

geo.add.edges and geo.remove.edges to interactively add or remove edges in a gGraph object.

getEdges to retrieve edges in different formats.

---

worldgraph *Worldwide geographic graphs*

Description

The datasets 'rawgraph.10k', 'rawgraph.40k', 'worldgraph.10k', and 'worldgraph.40k' are geographic graphs (gGraph objects) of the world, with respective resolutions of 10,242 and 40,962 vertices.

'rawgraph's are raw graphs as obtained directly from the method provided in references.

'worldgraph's are 'rawgraph's that have been modified manually to recitify connectivity between edges at some places. The most noticable change is that all edges oversea have been removed.

'globalcoord.10k' and 'globalcoord.40k' are matrices of geographic coordinates of nodes, used to construct 'worlgraph' objects.

'worldshape' is a shapefile of contries of the world (snapshot from 1994).

Usage

```
data(rawgraph.10k)
data(rawgraph.40k)
data(worldgraph.10k)
data(worldgraph.40k)
data(globalcoord.10k)
data(globalcoord.40k)
data(worldshape)
```

Format

worldgraph.10k and worldgraph.40k are gGraph objects with the following specificities:

@nodes.attr\$habitat habitat corresponding to each vertice; currently 'land' or 'sea'.

@meta\$color a matrix assigning a color for plotting vertices (second column) to different values of habitat (first column).

Source

Graph reconstructed by Andrea Manica.

References

=== On the construction of the graph ===
Randall, D. A.; Ringler, T. D.; Heikes, R. P.; Jones, P. & Baumgardner, J. Climate Modeling with Spherical Geodesic Grids *Computing in science & engineering*, 2002, **4**: 32-41.

Examples

```
data(worldgraph.10k)
worldgraph.10k

## plotting the object
plot(worldgraph.10k, reset=TRUE)
title("Hello world")

## zooming in
geo.zoomin(list(x=c(-12,45), y=c(33,75)))
title("Europe")
geo.zoomin(list(x=c(-12,2), y=c(50,60)))
plotEdges(worldgraph.10k)
title("United Kingdom")

## zooming out
# geo.zoomout() # needs clicking on device
geo.zoomin(list(x=c(-6,38), y=c(35,73)))
title("Europe")

## defining the subset of visible points
x <- worldgraph.10k[isInArea(worldgraph.10k)]
plot(x,reset=TRUE, edges=TRUE)
title("One subsetted object.")

## Not run:
## interactive zooming
geo.zoomin()

## End(Not run)
```

Index