Package ‘tbart’

February 20, 2015

Type Package

Title Teitz and Bart's p-Median Algorithm

Version 1.0

Date 2015-01-11

Author Chris Brunsdon

Maintainer Chris Brunsdon <christopher.brunsdon@nuim.ie>

Description Solves Teitz and Bart's p-median problem - given a set of points attempts to find subset of size p such that summed distances of any point in the set to the nearest point in p is minimised. Although generally effective, this algorithm does not guarantee that a globally optimal subset is found.

License GPL (>= 2)

Depends Rcpp (>= 0.10.3), sp

Suggests GISTools, RColorBrewer, rgeos

LinkingTo Rcpp

NeedsCompilation yes

Repository CRAN

Date/Publication 2015-02-13 20:23:02

R topics documented:

tbart-package *Teitz and Bart's p-median problem with Spatial* and Spatial*DataFrame objects*

Description

Solves Teitz and Bart's p-median problem - given a set of points attempts to find subset of size p such that summed distances of any point in the set to the nearest point in p is minimised. Although generally effective, this algorithm does not guarantee that a globally optimal subset is found.

Details

Package:	tbart
Type:	Package
Version:	1.0
Date:	2015-02-12
License:	GPL (>= 2)
Maintainer:	Chris Brunsdon mailto:christopher.brunsdon@nuim.ie

Author(s)

Chris Brunsdon

References

Teitz, M. B., and P. Bart (1968), Heuristic methods for estimating generalized vertex median of a weighted graph, Operations Research, 16, 955-961.

allocate *Teitz-Bart algorithm applied to Spatial* and Spatial*DataFrame objects*

Description

This function returns the allocations for each demand point - in terms of the index number of the record in `swdf2` assigned as the supply point. This version is useful as part of code inside other functions

Usage

```
allocate(swdf1, swdf2, force, p, metric, verbose = FALSE)
```

Arguments

`swdf1` - first Spatial* or Spatial*DataFrame objects

`swdf2` - second Spatial* or Spatial*DataFrame objects (if omitted, defaults to the same value as `swdf1`)

`force` - list of supply points or logical vector with length the same as the number of supply points that are forced to be used - eg existing outlets

`p` - either a guess at the initial p-median set of a single integer indicating the size of the set (which is then chosen randomly)

`metric` - the distance matrix (defaults to Euclidean computed via `euc.dists(swdf1,swdf2)` if not supplied)

`verbose` - if TRUE print out each swap in the algorithm (default is FALSE)

Value

List of nearest neigbour indices for each element from the p-median set

Examples

```
data(meuse)
coordinates(meuse) <- ~x+y
allocate(meuse,p=5)



require(RColorBrewer)
require(GISTools)
data(georgia)
allocations.list <- allocate(georgia2,p=5)
zones <- gUnaryUnion(georgia2,allocations.list)
plot(zones,col=brewer.pal(5,"Accent"))
plot(georgia2,border=rgb(0,0,0,0.1),add=TRUE)
points(coordinates(georgia2)[allocations.list,],pch=16,cex=2,col=rgb(1,0.5,0.5,0.1))
```

`allocations` *Teitz-Bart algorithm applied to Spatial* and Spatial*DataFrame objects*

Description

Return demand Spatial*Dataframe with new columns giving allocation id and distance to supply point

Usage

```
allocations(swdf1, swdf2, force, p, metric, verbose = FALSE)
```

Arguments

`swdf1`	- first Spatial* or Spatial*DataFrame objects
`swdf2`	- second Spatial* or Spatial*DataFrame objects (if omitted, defaults to the same value as `swdf1`)
`force`	- list of supply points or logical vector with length the same as the number of supply points that are forced to be used - eg e
`p`	- either a guess at the initial p-median set of a single integer indicating the size of the set (which is then chosen randomly)
`metric`	- the distance matrix (defaults to Euclidean computed via `euc.dists(swdf1,swdf2)` if not supplied)
`verbose`	- if TRUE print out each swap in the algorithm (default is FALSE)

Value

Copy of swdf1 with extra data columns called `allocation` and `allocdist` with indices for each element from the p-median set

Examples

```
require(RColorBrewer)
require(GISTools)
data(georgia)
georgia3 <- allocations(georgia2,p=5,force=c(1,120,44))
col.index <- match(georgia3$allocation,unique(georgia3$allocation))
col.alloc <- brewer.pal(5,'Accent')[col.index]
par(mfrow=c(1,2))
plot(georgia3,col=col.alloc)
choropleth(georgia3,georgia3$allocdist)


# Use in conjunction with rgeos
require(rgeos)
require(GISTools)
georgia3 <- allocations(georgia2,p=5,force=c(1,120,44))
georgia4 <- gUnaryUnion(georgia3,georgia3$allocation)
plot(georgia4)
plot(star.diagram(georgia3),col='darkred',lwd=2,add=TRUE)
```

`euc.dists` *Euclidean distances from a Spatial* or Spatial*DataFrame object*

Description

Euclidean distances from a Spatial* or Spatial*DataFrame object

Usage

```
euc.dists(swdf1, swdf2, scale)
```

Arguments

`swdf1` - First Spatial*DataFrame object

`swdf2` - Second Spatial*DataFrame object (if omitted, defaults to the same value as `swdf1`)

`scale` - allows re-scaling eg: value of 1000 means distances in km if coordinates of `swdf1`/`swdf2` in meters.

Value

Distance matrix (if `swdf1` or `swdf2` not SpatialPoints*, distances are based on points obtained from `coordinates` function)

Examples

```
data(meuse)
coordinates(meuse) <- ~x+y
euc.dists(meuse,scale=1000)
```

`mink.dists` *Minkowski distances from a Spatial* or Spatial*DataFrame object*

Description

Minkowski distances from a Spatial* or Spatial*DataFrame object

Usage

```
mink.dists(swdf1, swdf2, pwr, scale, weight)
```

Arguments

`swdf1` - First Spatial*DataFrame object

`swdf2` - Second Spatial*DataFrame object (if omitted, defaults to the same value as `swdf1`)

`pwr` - Minkowski exponent

`scale` - allows re-scaling eg: value of 1000 means distances in km if coordinates of `swdf1`/`swdf2` in meters.

`weight` - weight for each element in swdf1 (the demand locations)

Value

Distance matrix (if `swdf1` or `swdf2` not SpatialPoints*, distances are based on points obtained from `coordinates` function)

Examples

```
data(meuse)
coordinates(meuse) <- ~x+y
d1 <- mink.dists(meuse,pwr=1,scale=1000)   # Taxicab metric
d2 <- mink.dists(meuse,pwr=Inf,scale=1000) # Works for limiting case
```

`star.diagram` *Creates the lines for a ’star diagram’*

Description

Creates the lines for a ’star diagram’

Usage

```
star.diagram(swdf1, swdf2, alloc)
```

Arguments

`swdf1` - first Spatial* or Spatial*DataFrame objects

`swdf2` - second Spatial* or Spatial*DataFrame objects (if omitted, defaults to the same value as `swdf1`)

`alloc` - a list saying which coordinate in swdf2 is allocated to each point in swdf1 (if ommitted, looks for `allocation` column in `swdf1`)

Examples

```
data(meuse)
coordinates(meuse) <- ~x+y
allocations.list <- allocate(meuse,p=5)
star.lines <- star.diagram(meuse,alloc=allocations.list)
plot(star.lines)

# Acquire allocations from swdf1
require(GISTools)
set.seed(461976) # Reproducibility
data(georgia)
georgia3 <- allocations(georgia2,p=8)
plot(georgia3,border='grey')
plot(star.diagram(georgia3),col='darkblue',lwd=2,add=TRUE)
```

`tb` *Teitz-Bart algorithm applied to Spatial* and Spatial*DataFrame objects*

Description

This reports the p-median set

Usage

```
tb(swdf1, swdf2, p, metric, verbose = FALSE)
```

Arguments

`swdf1` - first Spatial* or Spatial*DataFrame objects - the 'demand' set

`swdf2` - second Spatial* or Spatial*DataFrame objects - the 'supply' set (if omitted, defaults to the same value as `swdf1`)

`p` - either a guess at the initial p-median set of a single integer indicating the size of the set (which is then chosen randomly)

`metric` - the distance matrix (defaults to Euclidean computed via `euc.dists(swdf1,swdf2)` if not supplied)

`verbose` - if TRUE print out each swap in the algorithm (default is FALSE)

Value

Set of point indices for p-median (may be local optimum)

Examples

```
data(meuse)
coordinates(meuse) <- ~x+y
tb(meuse,p=5)
```

`tb.raw` *Teitz-Bart algorithm applied to a 'raw' distance matrix*

Description

Teitz-Bart algorithm applied to a 'raw' distance matrix

Usage

```
tb.raw(d, guess, verbose = FALSE)
```

Arguments

`d` - A distance matrix (not necessarily Euclidean)

`guess` - a guess at the set of p points constituting the p-median

`verbose` - if TRUE print out each swap in the algorithm (default is FALSE)

Value

Set of point indices for p-median (may be local optimum)

Examples

```
x1 <- rnorm(100)
y1 <- rnorm(100)
d <- as.matrix(dist(cbind(x1,y1)))
tb.raw(d,c(1,2))
```

Index