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 .............................................. 2
allocate ................................................. 2
allocations ............................................... 3
euc.dists .................................................. 4
mink.dists .............................................. 5
star.diagram ........................................... 6
tb ........................................................... 7
tb ........................................................... 7
tb.raw .................................................... 7

Index 9
allocate

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

**Author(s)**

Chris Brunsdon

**References**


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)
allocations

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 neighbour indices for each element from the \( p \)-median set

Examples

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

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

Description

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

Usage

```r
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

```r
require(RColorBrewer)
require(GISTools)
data(georgia)
georgia3 <- allocations(georgia2, p=5, force=c(1,128,44))
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(1,128,44))
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)
mink.dists

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

```r
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)
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 omitted, 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)
georgia1 <- allocations(georgia2, p=8)
plot(georgia3, border='grey')
plot(star.diagram(georgia3), col='darkblue', lwd=2, add=TRUE)
Teitz-Bart algorithm applied to Spatial* and Spatial*DataFrame objects

Description

This reports the \( p \)-median set

Usage

\[
\text{tb}(\text{swdf1, swdf2, p, metric, verbose = FALSE})
\]

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 \text{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

\[
\text{data(meuse)}
\]
\[
\text{coordinates(meuse) <- ~x+y}
\]
\[
\text{tb(meuse, p=5)}
\]

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

Description

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

Usage

\[
\text{tb.raw(d, guess, verbose = FALSE})
\]
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

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

allocate, 2
allocations, 3

euc.dists, 4

mink.dists, 5

star.diagram, 6

tb, 7
tb.raw, 7
tbart-package, 2