Package 'ttbary'

February 19, 2022

Version 0.2-2

Date 2022-02-19

Title Barycenter Methods for Spatial Point Patterns

Maintainer Dominic Schuhmacher <dominic.schuhmacher@mathematik.uni-goettingen.de>

Depends R (>= 3.0.0), spatstat (>= 2.3-0)

LinkingTo Rcpp

Imports spatstat.geom, spatstat.core, spatstat.random, spatstat.linnet, grDevices, graphics, stats, Rcpp

Description Computes a point pattern in R^2 or on a graph that is representative of a collection of many data patterns. The result is an approximate barycenter (also known as Fréchet mean or prototype) based on a transport-transform metric. Possible choices include Optimal SubPattern Assignment (OSPA) and Spike Time metrics. Details can be found in Müller, Schuhmacher and Mateu (2020) <doi:10.1007/s11222-020-09932-y>.

LazyData yes

Encoding UTF-8

License GPL (>= 2)

RoxygenNote 7.1.2

NeedsCompilation yes

Author Raoul Müller [aut],
Dominic Schuhmacher [aut, cre]
(<https://orcid.org/0000-0001-7079-6313>)

Repository CRAN

Date/Publication 2022-02-19 13:20:02 UTC

R topics documented:

`kmeansbary` *Compute Pseudo-Barycenter of a List of Point Patterns*

Description

Starting from an initial candidate point pattern `zeta`, use a k-means-like algorithm to compute a local minimum in the barycenter problem based on the TT-2 metric for a list `pplist` of planar point patterns.

Usage

```
kmeansbary(
  zeta,
  pplist,
  penalty,
  add_del = Inf,
  surplus = 0,
  N = 200L,
  eps = 0.005,
  verbose = 0
)
```

Arguments

`zeta`	a point pattern. Object of class `ppp` or a list with components `x` and `y`.
`pplist`	a list of point patterns. Object of class `ppplist` or any list where each elements has components `x` and `y`.
`penalty`	the penalty for adding/deleting points when computing TT-2 distances.
`add_del`	for how many iterations shall the algorithm add points to / delete points from zeta if this is favorable? Defaults to Inf.
`surplus`	by how many points is the barycenter point pattern allowed to be larger than the largest input point pattern (among pplist and zeta) if add_del > 0. A larger number increases the computational load.
`N`	the maximum number of iterations.
`eps`	the algorithm stops if the relative improvement of the objective function between two iterations is less than eps.
`verbose`	the verbosity level. One of 0, 1, 2, 3, where 0 means silent and 3 means full details.

Details

Given k planar point patterns $\xi_1, \ldots, \xi_k$ (stored in `pplist`), this function finds a local minimizer ζ^* of

$$\sum_{j=1}^{k} \tau_2(\xi_j, \zeta)^2,$$

where τ_2 denotes the TT-2 metric based on the Euclidean metric between points.

Starting from an initial candidate point pattern `zeta`, the algorithm alternates between assigning a point from each pattern ξ_j to each point of the candidate and computing new candidate patterns by shifting, adding and deleting points. A detailed description of the algorithm is given in Müller, Schuhmacher and Mateu (2019).

For first-time users it is recommended to keep the default values and set `penalty` to a noticeable fraction of the diameter of the observation window, e.g. between 0.1 and 0.25 times this diameter.

Value

A list with components:

`cost`	the sum of squared TT-2 distances between the computed pseudo-barycenter and the point patterns.
`barycenter`	the pseudo-barycenter as a ppp-object.
`iterations`	the number of iterations required until convergence.

Author(s)

Raoul Müller <raoul.mueller@uni-goettingen.de>
Dominic Schuhmacher <schuhmacher@math.uni-goettingen.de>

References

Raoul Müller, Dominic Schuhmacher and Jorge Mateu (2020).
Metrics and Barycenters for Point Pattern Data.
Statistics and Computing 30, 953-972.
doi: 10.1007/s1122202009932y

See Also

`kmeansbaryeps` for a variant with epsilon relaxation that is typically faster

Examples

```
data(pplist_samecard)
plot(superimpose(pplist_samecard), cex=0.7, legend=FALSE,
     xlim=c(-0.2,1.2), ylim=c(-0.1,1.1), main="", use.marks=FALSE) #plotting the data

set.seed(12345)
zeta <- ppp(runif(100), runif(100))
plot(zeta, add=TRUE, col="beige", lwd=2, pch=16) #plotting the start-zeta over the data
```

```
res <- kmeansbary(zeta, pplist_samecard, penalty=0.1, add_del=Inf)
plot(res$barycenter, add=TRUE, col="blue", pch=16) #adding the computed barycenter in blue

res$cost
#[1] 30.30387
sumppdist(res$barycenter, pplist_samecard, penalty=0.1, type="tt", p=2, q=2)
#[1] 30.30387
#attr(,"distances")
#[1] 0.5991515 0.6133397 0.6040680 0.6020058 0.5648000 0.6415018 0.6385394 0.5784291 0.5985299
#[10] 0.6313200 0.7186154 ...
```

`kmeansbaryeps` *Compute Pseudo-Barycenter of a List of Point Patterns (with epsilon-relaxation)*

Description

Starting from an initial candidate point pattern `zeta`, use a k-means-like algorithm to compute a local minimum in the barycenter problem based on the TT-2 metric for a list `pplist` of planar point patterns.

Usage

```
kmeansbaryeps(
  epsvec,
  zeta,
  pplist,
  penalty,
  add_del = Inf,
  surplus = 0,
  relaxVec = c(20, 1, 1, 1),
  N = 200L,
  eps = 0.005,
  verbose = 0
)
```

Arguments

`epsvec`	a vector containing the values for the relaxed assignment. Last entry should be < 1/n, where n is the largest cardinality among the point patterns. Otherwise the algorithm has no guarantee of terminating in a local minimum! If epsvec[1] is too small, the computational load may be large. If in doubt, choose c(10^8,10^7,10^6,...,10/(n+1),1/(n+1)).
`zeta`	a point pattern. Object of class `ppp` or a list with components `x` and `y`.
`pplist`	a list of point patterns. Object of class `ppplist` or any list where each elements has components `x` and `y`.

penalty	the penalty for adding/deleting points when computing TT-2 distances.
add_del	for how many iterations shall the algorithm add points to / delete points from zeta if this is favorable? Defaults to Inf.
surplus	By how many points is the barycenter point pattern allowed to be larger than the largest input point pattern (among pplist and zeta) if add_del > 0. A larger number increases the computational load.
relaxVec	a vector of four integers controlling the epsilon-relaxation of the assignments. See details below.
N	the maximum number of iterations.
eps	the algorithm stops if the relative improvement of the objective function between two iterations is less than eps.
verbose	the verbosity level. One of 0, 1, 2, 3, where 0 means silent and 3 means full details.

Details

Given k planar point patterns $\xi_1, \ldots, \xi_k$ (stored in pplist), this function finds a local minimizer ζ^* of

$$\sum_{j=1}^{k} \tau_2(\xi_j, \zeta)^2,$$

where τ_2 denotes the TT-2 metric based on the Euclidean metric between points.

Starting from an initial candidate point pattern zeta, the algorithm alternates between assigning a point from each pattern ξ_j to each point of the candidate and computing new candidate patterns by shifting, adding and deleting points. A detailed description of the algorithm is given in Müller, Schuhmacher and Mateu (2019).

For first-time users it is recommended to keep the default values and set penalty to a noticeable fraction of the diameter of the observation window, e.g. between 0.1 and 0.25 times this diameter.

The argument relaxVec must be a vector of four integers c(a,b,c,d) > c(0,0,0,0). For the first a iterations step by step one entry of epsvec is additionally considered in the assignment, starting with only the first entry in the first iteration. In this a iterations the algorithm can stop if it has improved by less than eps between iterations. After a iterations all entries of epsvec before epsvec[b] are ignored and everytime the algorithm does not improve, the next d entries of epsvec are additionally considered in the following iterations. When the last entry of epsvec is considered in the assignments, the entries of epsvec before epsvec[c] are ignored. relaxVec defaults to c(20,1,1,1) meaning that in every one of the first 20 iterations one additional entry of epsvec is considered until the algorithm converges. This allows the algorithm to converge before the full epsvec was considered! For further details see example.

Warning: The argument relaxVec offers many different options for controlling the epsilon-relaxation of the assignments in order to save computation time. But choosing bad parameters may heavily increase the computational load! If in doubt, go with c(length(epsvec),1,1,1) (see examples).

Value

A list with components:

cost	the sum of squared TT-2 distances between the computed pseudo-barycenter and the point patterns.
barycenter	the pseudo-barycenter as a ppp-object.
iterations	the number of iterations required until convergence.

Author(s)

Raoul Müller <raoul.mueller@uni-goettingen.de>
Dominic Schuhmacher <schuhmacher@math.uni-goettingen.de>

References

Raoul Müller, Dominic Schuhmacher and Jorge Mateu (2020).
Metrics and Barycenters for Point Pattern Data.
Statistics and Computing 30, 953-972.
doi: 10.1007/s1122202009932y

See Also

kmeansbary for a similar function that works without epsilon relaxation

Examples

```
data(pplist_samecard)
plot(superimpose(pplist_samecard), cex=0.7, legend=FALSE,
     xlim=c(-0.2,1.2), ylim=c(-0.1,1.1), main="", use.marks=FALSE) #plotting the data

set.seed(12345)
zeta <- ppp(runif(100),runif(100))
plot(zeta, add=TRUE, col="beige", lwd=2, pch=16) #plotting the start-zeta over the data

epsvec <- c(1e8,1e7,1e6,1e5,1e4,1e3,1e2,10,1,10/101,1/101)

relaxVec1 <- c(length(epsvec),1,1,1)
#One additional entry of epsvec is considered in each iteration;
#algorithm can stop before full epsvec was used.
#Runs fast with little to no drawback in the quality of the computed solution.
#Time advantage more visible for large datasets.

relaxVec2 <- c(1,1,1,length(epsvec))
#In the first iteration only epsvec[1] is used, after that every assignment is exact.
#Not as fast as the previous version but usually no drawbacks at all in the computed solution.
#Time advantage more visible for large datasets.

relaxVec3 <- c(3,2,3,2)
#in the first 3 iterations epsvec[1],epsvec[1:2],epsvec[1:3] are used in the assignments,
#after that epsvec[2:x] is used, where x starts at 3 (=maximum(a,b)) and increases
#by 2 everytime the algorithm does not improve. When x >= length(epsvec) all assignments
#are done with epsvec[3:length(epsvec)].

res1 <- kmeansbaryeps(epsvec, zeta, pplist_samecard, penalty=0.1, add_del=5, relaxVec = relaxVec1)
```

```
res2 <- kmeansbaryeps(epsvec, zeta, pplist_samecard, penalty=0.1, add_del=5, relaxVec = relaxVec2)
res3 <- kmeansbaryeps(epsvec, zeta, pplist_samecard, penalty=0.1, add_del=5, relaxVec = relaxVec3)
plot(res1$barycenter, add=TRUE, col="blue", pch=16) #adding the computed barycenter in blue
```

`kmeansbarynet` *Compute Pseudo-Barycenter of a List of Point Patterns on a Network*

Description

Starting from an initial candidate point pattern `zeta`, use a k-means-like algorithm to compute a local minimum in the barycenter problem based on the TT-1 metric for a collection of point patterns on a network. The data needs to be in a special form which can be produced with the function `netsplit`.

Usage

```
kmeansbarynet(dpath, zeta, ppmatrix, penalty, N = 200L, eps = 0.005)
```

Arguments

`dpath`	a square matrix whose (`i`,`j`)th entry is the shortest-path distance between vertex `i` and vertex `j`. Vertex means either network vertex or data point.
`zeta`	a vector containing the vertex-indices of the initial candidate for the barycenter.
`ppmatrix`	a matrix specifying in its columns the vertex-indices of the different data point patterns. A virtual index that is one greater than the maximum vertex-index can be used to fill up columns so they all have the same length (see examples).
`penalty`	the penalty for adding/deleting points when computing TT-1 distances.
`N`	the maximum number of iterations.
`eps`	the algorithm stops if the relative improvement of the objective function between two iterations is less than eps.

Details

Given k planar point patterns $\xi_1, \ldots, \xi_k$ (specified by giving the indices of their points in the k columns of `ppmatrix`), this function finds a local minimizer ζ^* of

$$\sum_{j=1}^{k} \tau_1(\xi_j, \zeta),$$

where τ_1 denotes the TT-1 metric based on the shortest-path metric between points in the network.

Starting from an initial candidate point pattern `zeta` (specified by giving the indices of its points), the algorithm alternates between assigning a point from each pattern ξ_j to each point of the candidate and computing new candidate patterns by shifting points (addition and deletion of points is currently not implemented). A detailed description of the algorithm is given in Müller, Schuhmacher and Mateu (2019).

The most convenient way to obtain objects dpath and ppmatrix of the right form is by calling netsplit and extracting components network$dpath and ppmatrix from the resulting object (see examples below).

Value

A list containing the following components:

cost	the sum of TT-1 distances between the computed pseudo-barycenter and the point patterns.
barycenter	the pseudo-barycenter as a vector of vertex-indices.
zetalist	a list containing the alternative vertex-indices for each point of the pseudo-barycenter.
barycost	a vector containing the cluster costs for each point of the pseudo-barycenter (the alternative indices in zetalist lead to the same cluster cost).
perm	the permutation matrix for the clusters.
iterations	the number of iterations required until convergence.

Author(s)

Raoul Müller <raoul.mueller@uni-goettingen.de>
Dominic Schuhmacher <schuhmacher@math.uni-goettingen.de>

References

Raoul Müller, Dominic Schuhmacher and Jorge Mateu (2020).
Metrics and Barycenters for Point Pattern Data.
Statistics and Computing 30, 953-972.
doi: 10.1007/s1122202009932y

See Also

kmeansbary for a similar function for point patterns in R^2

Examples

```
set.seed(123456)
nvert <- 100 #number of vertices in the network
npp <- 5 #number of data point patterns
npts <- 40 #number of points per data point pattern
ln <- delaunayNetwork(runifpoint(nvert)) #create an artificial network
ppnetwork <- runiflpp(npts,ln,nsim = npp)
  #simulate npp point patterns with npts points each

plot(ppnetwork[[1]]$domain, cex=0.5, main="")
for (i in 1:npp) {
  plot(as.ppp(ppnetwork[[i]]),vpch=1,col=i,add=TRUE)
     #plotting the point patterns in different colors
}
```

```
res <- netsplit(ln, ppnetwork)
  #incorporate data point patterns into the network
  #calculating all pairwise distances between vertices
  #and creating matrix of vertex-indices of data point patterns

zeta <- sample(res$nvirtual - 1, median(res$dimensions))
  #sample random vertex-indices in the network
  #taking as cardinality the median of point pattern cardinalities

res2 <- kmeansbarynet(res$network$dpath, zeta, res$ppmatrix, penalty = 0.1)

barycenter <- ppp(res$network$vertices$x[res2$barycenter], res$network$vertices$y[res2$barycenter])
  #construct the barycenter pattern based on the index information in res2
points(barycenter,cex = 1.2, lwd = 2, pch = 4, col = "magenta")
  #add the computed barycenter as magenta crosses

res2$cost
#[1] 18.35171
sumppdistnet(res$network$dpath, res2$barycenter, res$ppmatrix, penalty=0.1, type="tt", p=1, q=1)
#[1] 18.35171
#attr(,"distances")
#[1] 3.666471 3.774709 3.950079 3.841166 3.119284
```

`kmeansbaryweightnet` *Compute weighted Pseudo-Barycenter of a List of Point Patterns on a Network*

Description

Starting from an initial candidate point pattern `zeta`, use a k-means-like algorithm to compute a local minimum in the barycenter problem based on the TT-1 metric for a collection of point patterns on a network. The data needs to be in a special form which can be produced with the function `netsplit`.

Usage

```
kmeansbaryweightnet(
  dpath,
  zeta,
  ppmatrix,
  weights,
  penalty,
  N = 200L,
  eps = 0.005
)
```

Arguments

dpath	a square matrix whose (i,j)th entry is the shortest-path distance between vertex i and vertex j. Vertex means either network vertex or data point.
zeta	a vector containing the vertex-indices of the initial candidate for the barycenter.
ppmatrix	a matrix specifying in its columns the vertex-indices of the different data point patterns. A virtual index that is one greater than the maximum vertex-index can be used to fill up columns so they all have the same length (see examples).
weights	a vector with weights for each point pattern
penalty	the penalty for adding/deleting points when computing TT-1 distances.
N	the maximum number of iterations.
eps	the algorithm stops if the relative improvement of the objective function between two iterations is less than eps.

Details

Given k planar point patterns $\xi_1, \ldots, \xi_k$ (specified by giving the indices of their points in the k columns of ppmatrix), this function finds a local minimizer ζ^* of

$$\sum_{j=1}^{k} \tau_1(\xi_j, \zeta),$$

where τ_1 denotes the TT-1 metric based on the shortest-path metric between points in the network.

Starting from an initial candidate point pattern zeta (specified by giving the indices of its points), the algorithm alternates between assigning a point from each pattern ξ_j to each point of the candidate and computing new candidate patterns by shifting points (addition and deletion of points is currently not implemented). A detailed description of the algorithm is given in Müller, Schuhmacher and Mateu (2019).

The most convenient way to obtain objects dpath and ppmatrix of the right form is by calling netsplit and extracting components network$dpath and ppmatrix from the resulting object (see examples below).

Value

A list containing the following components:

cost	the sum of TT-1 distances between the computed pseudo-barycenter and the point patterns.
barycenter	the pseudo-barycenter as a vector of vertex-indices.
zetalist	a list containing the alternative vertex-indices for each point of the pseudo-barycenter.
barycost	a vector containing the cluster costs for each point of the pseudo-barycenter (the alternative indices in zetalist lead to the same cluster cost).
perm	the permutation matrix for the clusters.
iterations	the number of iterations required until convergence.

Author(s)

Raoul Müller <raoul.mueller@uni-goettingen.de>
Dominic Schuhmacher <schuhmacher@math.uni-goettingen.de>

References

Raoul Müller, Dominic Schuhmacher and Jorge Mateu (2020).
Metrics and Barycenters for Point Pattern Data.
Statistics and Computing 30, 953-972.
doi: 10.1007/s1122202009932y

See Also

kmeansbary for a similar function for point patterns in R^2

Examples

```
set.seed(123456)
nvert <- 100 #number of vertices in the network
npp <- 5 #number of data point patterns
npts <- 40 #number of points per data point pattern
ln <- delaunayNetwork(runifpoint(nvert)) #create an artificial network
ppnetwork <- runiflpp(npts,ln,nsim = npp)
  #simulate npp point patterns with npts points each

plot(ppnetwork[[1]]$domain, cex=0.5, main="")
for (i in 1:npp) {
  plot(as.ppp(ppnetwork[[i]]),vpch=1,col=i,add=TRUE)
     #plotting the point patterns in different colors
}

res <- netsplit(ln, ppnetwork)
  #incorporate data point patterns into the network
  #calculating all pairwise distances between vertices
  #and creating matrix of vertex-indices of data point patterns

zeta <- sample(res$nvirtual - 1, median(res$dimensions))
  #sample random vertex-indices in the network
  #taking as cardinality the median of point pattern cardinalities

res2 <- kmeansbaryweightnet(res$network$dpath, zeta, res$ppmatrix,
                            weights = c(1,2,3,2,1), penalty = 0.1)

barycenter <- ppp(res$network$vertices$x[res2$barycenter], res$network$vertices$y[res2$barycenter])
  #construct the barycenter pattern based on the index information in res2
points(barycenter,cex = 1.2, lwd = 2, pch = 4, col = "magenta")
  #add the computed barycenter as magenta crosses

res2$cost
#[1] 18.35171
sumppdistnet(res$network$dpath, res2$barycenter, res$ppmatrix, penalty=0.1, type="tt", p=1, q=1)
```

```
#[1] 18.35171
#attr(,"distances")
#[1] 3.666471 3.774709 3.950079 3.841166 3.119284
```

`netsplit` *Incorporate Point Patterns into a Network*

Description

Given a network and a list of point patterns on this network, create a new network from all the vertices of the original network plus all the points in the patterns, splitting any edges that contain such points into several shorter edges. This function keeps track which vertex-indices represent each of the data point patterns. The returned object contains all the components needed for a call to `kmeansbarynet`.

Usage

```
netsplit(network, pplist)
```

Arguments

`network`	an object of class `linnet` or `lpp`. In the latter case the `domain` component is extracted and any points of the `lpp` are ignored.
`pplist`	a list containing (at least) `x`- and `y`-coordinates of the point patterns, which will be projected onto the network

Details

This function relies heavily on code from the package `spatstat` to create the new network and efficiently compute all pairwise shortest-path distances between the new vertices.

If not all point patterns are of the same size, this function fills up the vertex-indices of the smaller patterns with a virtual index that is one larger than the maximal index appearing in the new network. This structure is required for calling `kmeansbarynet`.

Value

A list containing the following components:

`network`	the new network with all the points added as vertices. Contains also the matrix of shortest-path distances between all these points.
`ppmatrix`	a matrix containing the new vertex-indices of the data point patterns, one column corresponds to one point pattern.
`dimensions`	a vector containing the cardinalities of the data point patterns.
`nvirtual`	the index of the virtual point.

Author(s)

Raoul Müller <raoul.mueller@uni-goettingen.de>
Dominic Schuhmacher <schuhmacher@math.uni-goettingen.de>

See Also

kmeansbarynet

Examples

```
# See the example for kmeansbarynet.
```

---

plotmatch *Plot Optimal Matching between Two Point Patterns*

Description

After calling ppdist with argument ret_matching = TRUE in a situation where it makes sense to assign to the points of the patterns ξ and η coordinates in R^2, this function may be used to display the result graphically.

Usage

```
plotmatch(
  xi,
  eta,
  dmat,
  res,
  penalty,
  p = 1,
  cols = c(2, 4),
  pchs = c(1, 1),
  cexs = c(1, 1),
  ...
)
```

Arguments

xi, eta	objects of class ppp.
dmat	a matrix specifying in its (i, j)-th entry the distance from the i-th point of ξ to the j-th point of η.
res	the object returned by the call to ppdist with ret_matching = TRUE.
penalty	a positive number. The penalty for adding/deleting points.
p	a number > 0. The order of the TT- or RTT-distance computed.

`cols, pchs, cexs`
: vectors of length 2 specifying the corresponding graphic parameters col, pch and cex for plotting the two point patterns.

`...`
: further graphic parameters passed to the code that draws the line segments between the points.

Details

The default use-case is to plot a matching obtained with `ppdist`. In that case `dmat`, `penalty` and `p` should be the same as in the call to `ppdist`. These objects are used to display additional information about the matching.

Value

Used for the side effect of plotting.

Author(s)

Dominic Schuhmacher <schuhmacher@math.uni-goettingen.de>

See Also

`ppdist`

Examples

```
# See examples for ppdist
```

---

ppdist *Compute Distance Between Two Point Patterns*

Description

Based on an arbitrary matrix of "distances" between the points of two point patterns ξ and η, this function computes versions of the transport-transform distance between ξ and η.

Usage

```
ppdist(
  dmat,
  penalty = 1,
  type = c("tt", "rtt", "TT", "RTT"),
  ret_matching = FALSE,
  p = 1,
  precision = NULL
)
```

Arguments

`dmat`	a matrix specifying in its (i, j)-th entry the distance from the i-th point of ξ to the j-th point of η.
`penalty`	a positive number. The penalty for adding/deleting points.
`type`	either `"tt"`/`"TT"` for the transport-transform metric or `"rtt"`/`"RTT"` for the relative transport-transform metric.
`ret_matching`	logical. Shall the optimal point matching be returned?
`p`	a number > 0. The matching is chosen such that the p-th order sum (ℓ_p-norm) is minimized.
`precision`	a small positive integer value. The precisions of the computations, which are currently performed in integers. After correcting for the penalty, `dmat^p` is divided by its largest entry, multiplied by `10^precision` and rounded to compute the optimal matching. The default value `NULL` chooses maximal integer precision possible, which is `precision = 9` on almost all systems.

Details

The transport-transform (TT) distance gives the minimal total cost for "morphing" the pattern ξ into the pattern η by way of shifting points (at costs specified in `dmat`) and adding or deleting points (each at cost `penalty`). The total cost is determined as

$$\left(\sum_{j=1}^{n} c_j^p\right)^{1/p},$$

where c_j denotes the cost for the jth individual operation and n is the cardinality of the larger point pattern.

The relative transport-transform (RTT) metric is exactly the same, but the sum in the total cost is divided by the larger cardinality:

$$\left(\frac{1}{n}\sum_{j=1}^{n} c_j^p\right)^{1/p}.$$

The TT- and RTT-metrics form an umbrella concept that includes the OSPA and Spike Time metrics frequently used in the literature. See Müller, Schuhmacher and Mateu (2019) for details.

Value

The corresponding distance between the point patterns if `ret_matching` is `FALSE`.

Otherwise a list with components `dist` containing this distance and two vectors `target1,target2` of integers, where `target`i specifies the indices of the points in the other pattern that the points of the i-th pattern are matched to and `NA` every time a point is deleted.

There may be a minus in front of an index, where `-j` indicates that the corresponding pairing with point `j` would be over a distance of more than $2^{1/p} \cdot$ `penalty`. This is equivalent to saying that the corresponding point of the first pattern is deleted and the j-th point of the second pattern is added.

Note that having more than one minus implies that the matching is non-unique.

Author(s)

Dominic Schuhmacher <schuhmacher@math.uni-goettingen.de>

References

Raoul Müller, Dominic Schuhmacher and Jorge Mateu (2020).
Metrics and Barycenters for Point Pattern Data.
Statistics and Computing 30, 953-972.
doi: 10.1007/s1122202009932y

Examples

```
# small example
# -------------
set.seed(181230)
xi <- spatstat.random::rpoispp(20)
eta <- spatstat.random::rpoispp(20)
dmat <- spatstat.geom::crossdist(xi,eta)
res <- ppdist(dmat, penalty=1,  type="rtt", ret_matching=TRUE, p=1)
plotmatch(xi, eta, dmat, res, penalty=1, p=1)
res$dist

# for comparison: ospa-distance computation from spatstat:
res_ospa <- spatstat.geom::pppdist(xi,eta,"spa")
res_ospa$distance  # exactly the same as above because nothing gets cut off


# same example, but with a much smaller penalty for adding/deleting points
# -------------------------------------------------------------
res <- ppdist(dmat, penalty=0.1,  type="rtt", ret_matching=TRUE, p=1)
plotmatch(xi, eta, dmat, res, penalty=0.1, p=1)
  # dashed lines indicate points that are deleted and re-added at new position
  # grey segments on dashed lines indicate the cost of deletion plus re-addition
res$dist

# for comparison: ospa-distance computation from spatstat
# (if things do get cut off, we have to ensure that the cutoff distances
# are the same, thus cutoff = 2^(1/p) * penalty):
res_ospa <- spatstat.geom::pppdist(xi,eta,"spa",cutoff=0.2)
res_ospa$distance  # NOT the same as above
res_ospa$distance - abs(xi$n-eta$n) * 0.1 / max(xi$n,eta$n)  # the same as above


# a larger example
# ----------------------------------------------------------------
set.seed(190203)
xi <- spatstat.random::rpoispp(2000)
eta <- spatstat.random::rpoispp(2000)
dmat <- spatstat.geom::crossdist(xi,eta)
res <- ppdist(dmat, penalty = 0.1,  type = "rtt", ret_matching = TRUE, p = 1)
res$dist
# takes about 2-3 seconds
```

ppdistnet *Compute Distance Between Two Point Patterns on a Network*

Description

Based on an arbitrary matrix of "distances" on a network, this function computes versions of the transport-transform distance between two point patterns ξ and η on this network.

Usage

```
ppdistnet(
  dmat,
  xi = NULL,
  eta = NULL,
  penalty = 1,
  type = c("tt", "rtt", "TT", "RTT"),
  ret_matching = FALSE,
  p = 1,
  precision = NULL
)
```

Arguments

dmat	a matrix specifying in its (i, j)-th entry the shortest-path distance from the i-th point of ξ to the j-th point of η OR the distance matrix of a whole network. In the latter case arguments ξ and η have to be specified.
xi	a vector specifying the vertex-indices of ξ, only needed if dmat is the distance matrix of a whole network.
eta	a vector specifying the vertex-indices of η, only needed if dmat is the distance matrix of a whole network.
penalty	a positive number. The penalty for adding/deleting points.
type	either "tt"/"TT" for the transport-transform metric or "rtt"/"RTT" for the relative transport-transform metric.
ret_matching	Logical. Shall the optimal point matching be returned?
p	a number > 0. The matching is chosen such that the p-th order sum (ℓ_p-norm) is minimized.
precision	a small positive integer value. The precision of the computations, which are currently performed in integers. After correcting for the penalty, dmat^p is divided by its largest entry, multiplied by 10^precision and rounded to compute the optimal matching. The default value NULL chooses maximal integer precision possible, which is precision = 9 on almost all systems.

Details

This function provides a more convenient way for computing (relative) transport-transform distances on networks if the points of the patterns are given in terms of indices of network vertices. If `dmat` contains only the distances between the points of ξ and η, this function does the same as `ppdist`.

Value

The corresponding distance between the point patterns if `ret_matching` is `FALSE`.

Otherwise a list with components `dist` containing this distance and two vectors `target1`, `target2` of integers, where `target`i specifies the indices of the points in the other pattern that the points of the i-th pattern are matched to and `NA` every time a point is deleted.

There may be a minus in front of an index, where `-j` indicates that the corresponding pairing with point `j` would be over a distance of more than $2^{1/p} \cdot$ `penalty`. This is equivalent to saying that the corresponding point of the first pattern is deleted and the j-th point of the second pattern is added.

Note that having more than one minus implies that the matching is non-unique.

Author(s)

Raoul Müller <raoul.mueller@uni-goettingen.de>
Dominic Schuhmacher <schuhmacher@math.uni-goettingen.de>

See Also

`ppdist`

Examples

```
set.seed(123456)
nvert <- 100 #number of vertices in the network
lambda <- 0.5 #expected number of points per unit length
ln <- delaunayNetwork(runifpoint(nvert)) #create an artificial network
ppnetwork <- rpoislpp(lambda, ln, nsim = 2)
  #simulate two point patterns on the network

plot(ppnetwork[[1]]$domain, cex=0.5, main="")
plot(as.ppp(ppnetwork[[1]]),vpch=1,col=2,add=TRUE)
plot(as.ppp(ppnetwork[[2]]),vpch=1,col=4,add=TRUE)

res <- netsplit(ln, ppnetwork)
  #incorporate data point patterns into the network
  #calculating all pairwise distances between vertices
  #and creating matrix of vertex-indices of data point patterns

xi <- res$ppmatrix[1:npoints(ppnetwork[[1]]), 1]
eta <- res$ppmatrix[1:npoints(ppnetwork[[2]]), 2]
res2 <- ppdistnet(res$network$dpath, xi = xi, eta = eta,
                  penalty = 1, type = "tt", ret_matching = TRUE, p = 1)
res2
```

pplist-data *Simulated Point Pattern Lists*

Description

Lists of simulated point patterns for illustrating the computation of barycenters.

Usage

```
pplist_samecard

pplist_diffcard
```

Format

Objects of class `pplist`, which are essentially lists of ppp-objects.

An object of class `ppplist` (inherits from `solist`, `anylist`, `listof`, `list`) of length 80.

An object of class `ppplist` (inherits from `solist`, `anylist`, `listof`, `list`) of length 50.

Details

`pplist_samecard` contains 80 point patterns of 100 points each. The patterns were independently generated from a distribution that creates quite distinctive clusters.

`pplist_diffcard` contains 50 point patterns with cardinalities ranging from 17 to 42. The patterns were independently generated from a distribution that creates overlapping clusters.

Author(s)

Raoul Müller <raoul.mueller@uni-goettingen.de>
Dominic Schuhmacher <schuhmacher@math.uni-goettingen.de>

Examples

```
# plot the first eight patterns of each data set
plot(superimpose(pplist_samecard[1:8]), legend=FALSE, cex=0.4, cols=1:8)
plot(superimpose(pplist_diffcard[1:8]), legend=FALSE, cex=0.4, cols=1:8)
```

sumppdist *Compute Sum of q-th Powers of Distances Between a Point Pattern and a List of Point Patterns*

Description

Determine the Euclidean distance based TT-p-distances (or RTT-p-distances) between a single point pattern `zeta` and each point pattern in a list `pplist`. Then compute the sum of q-th powers of these distances.

Usage

```
sumppdist(
  zeta,
  pplist,
  penalty = 1,
  type = c("tt", "rtt", "TT", "RTT"),
  p = 1,
  q = 1
)
```

Arguments

`zeta`	an object of class `ppp`.
`pplist`	an object of class `pppplist` or an object that can be coerced to this class, such as a list of `ppp` objects.
`penalty`	a positive number. The penalty for adding/deleting points.
`type`	either `"tt"`/`"TT"` for the transport-transform metric or `"rtt"`/`"RTT"` for the relative transport-transform metric.
`p`	a number > 0. Matchings between `zeta` and the patterns in `pplist` are chosen such that the p-th order sums (ℓ_p-norms) of the Euclidean distances are minimized.
`q`	a number > 0.

Details

The main purpose of this function is to evaluate the relative performance of approximate q-th order barycenters of point patterns. A true q-th order barycenter of the point patterns $\xi_1, \ldots, \xi_k$ with respect to the TT-p metric τ_p minimizes

$$\sum_{j=1}^{k} \tau_p(\xi_j, \zeta)^q$$

in ζ.

The most common choices are `p = q = 1` and `p = q = 2`. Other choices have not been tested.

Value

A nonnegative number, the q-th order sum of the TT-p- or RTT-p-distances between `zeta` and each pattern in `pplist`. This number has an attribute `distances` that contains the individual distances.

Author(s)

Dominic Schuhmacher <schuhmacher@math.uni-goettingen.de>

See Also

`ppdist` for computation of TT-p- and RTT-p-metrics,
`kmeansbary` for finding a local minimum of the above sum for `p = q = 2`

Examples

```
# See the examples for kmeansbary
```

`sumppdistnet` *Compute Sum of q-th Powers of Distances Between a Point Pattern and a Collection of Point Patterns on a Network*

Description

Based on the shortest-path metric in a network, determine the TT-p-distances (or RTT-p-distances) between a single point pattern `zeta` and a collection of point patterns. Then compute the sum of q-th powers of these distances. The point patterns are specified by vectors of indices referring to the vertices in the network.

Usage

```
sumppdistnet(
  dmat,
  zeta,
  ppmatrix,
  penalty = 1,
  type = c("tt", "rtt", "TT", "RTT"),
  p = 1,
  q = 1
)
```

Arguments

`dmat`	the distance matrix of a network containing all shortest-path distances between its vertices.
`zeta`	a vector specifying the vertex-indices of zeta.

ppmatrix	a matrix specifying in its columns the vertex-indices of the point patterns in the collection. A virtual index that is one greater than the maximum vertex-index in the network can be used to fill up columns so that they all have the same length.
penalty	a positive number. The penalty for adding/deleting points.
type	either "tt"/"TT" for the transport-transform metric or "rtt"/"RTT" for the relative transport-transform metric.
p	a number > 0. Matchings between zeta and the patterns in ppmatrix are chosen such that the p-th order sums (ℓ_p-norms) of the shortest-path distances are minimized.
q	a number > 0.

Details

The main purpose of this function is to evaluate the relative performance of approximate q-th order barycenters of point patterns. A true q-th order barycenter of the point patterns $\xi_1, \ldots, \xi_k$ with respect to the TT-p metric τ_p minimizes

$$\sum_{j=1}^{k} \tau_p(\xi_j, \zeta)^q$$

in ζ.

The most common choices are p = q = 1 and p = q = 2. Other choices have not been tested.

Value

A nonnegative number, the q-th order sum of the TT-p- or RTT-p-distances between the patterns represented by zeta and ppmatrix. This number has an attribute distances that contains the individual distances.

Author(s)

Raoul Müller <raoul.mueller@uni-goettingen.de>
Dominic Schuhmacher <schuhmacher@math.uni-goettingen.de>

See Also

kmeansbarynet, sumppdist

Examples

```
# See examples for kmeansbarynet
```

Index