# Package 'transport'

October 3, 2017

**Version** 0.9-4

**Date** 2017-10-03

**Title** Optimal Transport in Various Forms

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

**Depends** R (>= 3.0.0)

**Imports** grDevices, graphics, stats, Rcpp (>= 0.12.10)

**Suggests** animation, ks

**LinkingTo** Rcpp

**Description** Solve optimal transport problems. Compute Wasserstein distances (a.k.a. Kantorovitch, Fortet--Mourier, Mallows, Earth Mover's, or minimal L_p distances), return the corresponding transference plans, and display them graphically. Objects that can be compared include grey-scale images, (weighted) point patterns, and mass vectors.

**LazyData** yes

**Encoding** UTF-8

**License** GPL (>= 2)

**URL** http://www.dominic.schuhmacher.name

**RoxygenNote** 6.0.1

**NeedsCompilation** yes

**Author** Dominic Schuhmacher [aut, cre],
Björn Bähre [aut] (aha and power diagrams),
Carsten Gottschlich [aut] (simplex and shortlist),
Florian Heinemann [aut] (transport_track),
Bernhard Schmitzer [aut] (shielding),
Timo Wilm [ctb] (wpp)

**Repository** CRAN

**Date/Publication** 2017-10-03 20:00:10 UTC

# R topics documented:



---

`transport-package` *Optimal Transport in Various Forms*

---

## Description

Solve optimal transport problems. Compute Wasserstein distances (a.k.a. Kantorovitch, Fortet–Mourier, Mallows, Earth Mover's, or minimal $L_p$ distances), return the corresponding transport plans, and display them graphically. Objects that can be compared include grey-scale images, (weighted) point patterns, and mass vectors.

## Details

| | |
|---|---|
| Package: | transport |
| Type: | Package |
| Version: | 0.9-4 |
| Date: | 2017-10-03 |
| License: | GPL (>=2) |
| LazyData: | yes |

The main end-user function is transport. It computes optimal transport plans between images (class pgrid), point patterns (class pp), weighted point patterns (class wpp) and mass vectors, based on various algorithms. These transport plans can be plotted. The function wasserstein allows for the numerical computation of $p$-th order Wasserstein distances.

Most functions in this package are designed for data in two and higher dimensions. A quick tool for computing the $p$-th order Wasserstein distance between univariate samples is wasserstein1d.

## Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>
Björn Bähre <bjobae@gmail.com> (code for aha-method)
Carsten Gottschlich <gottschlich@math.uni-goettingen.de>
__(original java code for shortlist- and revsimplex-methods)
Florian Heinemann <florian.heinemann@stud.uni-goettingen.de> (transport_track-function)
Bernhard Schmitzer <schmitzer@uni-muenster.de> (shielding-method)

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

## References

See help page for the function transport.

## Examples

```
##  See examples for function transport
```

---

# aha *Solve Transportation Problem by Aurenhammer–Hoffmann–Aronov Method*

---

## Description

Solve transportation problem by Aurenhammer–Hoffmann–Aronov Method.

## Usage

```
aha(a, b, nscales = 1, scmult = 2, factr = 1e+05, maxit = 10000, powerdiag=FALSE,
    wasser = FALSE, wasser.spt = NA, approx=FALSE, ...)
transport_apply(a, tplan)
transport_error(a, b, tplan)
```

## Arguments

| | |
|---|---|
| a | an $m \times n$ matrix. a is treated as a measure on $[0, m] \times [0, n]$ with constant density on each unit square $[i, i+1) \times [j, j+1)$. |
| b | either a matrix such that dim(a)==dim(b) and sum(a)==sum(b) or a list of three vectors of equal length, named x, y and mass such that sum(a)==sum(b$mass), representing a discrete measure on $[0, m]) \times [0, n]$. |

| | |
|---|---|
| tplan | a transference plan from a (to b), typically an optimal transference plan obtained by a call to aha. |
| nscales, scmult | the number of scales to use for the multiscale approach (the default is 1 meaning no multiscale approach), and the factor by which the number of pixels in each dimension is multiplied to get from a coarser to the next finer scale. |
| factr, maxit | parameters passed to the underlying L-BFGS-B algorithm (via the argument control in the R-function optim). |
| powerdiag | logical. Instead of an optimal transference plan, should the parameters for the optimal power diagram be returned? |
| wasser | logical. Instead of an optimal transference plan, should the $L_2$-Wasserstein-distance between a and b be returned directly? |
| wasser.spt | the number of support points used to approximate the discrete measure b. Defaults to NA meaning the full set of support points of b is used. If this argument is not NA, wasser is set to TRUE. |
| approx | logical. If TRUE, an approximation to the objective function is used during optimization. |
| ... | further arguments passed to optim via its argument control. |

## Details

The function aha implements the algorithm by Aurenhammer, Hoffmann and Aronov (1998) for finding optimal transference plans in terms of the squared Euclidean distance in two dimensions. It follows the more detailed description given in Mérigot (2011) and also implements the multiscale version presented in the latter paper.

The functions transport_apply and transport_error serve for checking the accuracy of the transference plan obtained by aha. Since this transference plan is obtained by continuous optimization it will not transport exactly to the measure b, but to the measure transport_apply(a, tplan). By transport_error(a, b, tplan) the sum of absolut errors between the transported a-measure and the b-measure is obtained.

## Value

If powerdiag and wasser are both FALSE, a data frame with columns from, to and mass, which specify from which knot to which other knot what amount of mass is sent in the optimal transference plan. Knots are given as indices in terms of the usual column major enumeration of the matrices a and b. There are plot methods for the classes pgrid and pp, which can plot this solution.

If powerdiag is TRUE and wasser is FALSE, a list with components xi, eta, w and rect, which specify the parameters for the optimal power diagram in the same format as needed for the function power_diagram. Note that rect is always c(0,m,0,n).

If wasser is TRUE, a data frame with columns wasser.dist and error.bound of length one, where error.bound gives a bound on the absolute error in the Wasserstein distance due to approximating the measure b by a measure on a smaller number of support points.

## Author(s)

Björn Bähre <bjobae@gmail.com>
(slightly modified by Dominic Schuhmacher <dschuhm1@uni-goettingen.de>)

## References

F. Aurenhammer, F. Hoffmann and B. Aronov (1998). Minkowski-type theorems and least-squares clustering. Algorithmica 20(1), 61–76.

Q. Mérigot (2011). A multiscale approach to optimal transport. Eurographics Symposium on Geometry Processing 30(5), 1583–1592.

## See Also

transport, which is a convenient wrapper function for various optimal transportation algorithms.

## Examples

```
res <- aha(random32a$mass, random32b$mass)
plot(random32a, random32b, res, lwd=0.75)

aha(random64a$mass, random64b$mass, nscales=3, scmult=5, wasser.spt=512, approx=TRUE)
```

---

# all.equal (transport objects)

*Methods for Judging Near Equality of Objects of Class pgrid, pp or wpp*

---

## Description

Methods for judging near equality of objects of class pgrid or pp or wpp

## Usage

```
## S3 method for class 'pgrid'
all.equal(target, current, ...)
## S3 method for class 'pp'
all.equal(target, current, ...)
## S3 method for class 'wpp'
all.equal(target, current, ...)
```

## Arguments

| | |
|---|---|
| target, current | the objects of the same class to be compared. |
| ... | currently without effect. |

## Value

Either TRUE or a vector of mode "character" describing the differences between target and current.

## Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>

## See Also

all.equal (base), compatible

---

compatible *Test whether Two Objects are Compatible*

---

## Description

Test whether two objects of the same class are 'of similar shape' so that the function transport can be applied.

## Usage

```
compatible(target, current, ...)
## S3 method for class 'pgrid'
compatible(target, current, ...)
## S3 method for class 'pp'
compatible(target, current, ...)
## S3 method for class 'wpp'
compatible(target, current, ...)
```

## Arguments

| | |
|---|---|
| target, current | to objects of the same class to be compared. |
| ... | currently without effect. |

## Value

Logical.

## Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>

## See Also

all.equal

## methods *Print and Summary Methods for Objects of Class pgrid, pp and wpp*

### Description

Prints a brief description of a pixel grid or a point pattern.

### Usage

```
## S3 method for class 'pgrid'
print(x, ...)
## S3 method for class 'pp'
print(x, ...)
## S3 method for class 'wpp'
print(x, ...)
## S3 method for class 'pgrid'
summary(object, ...)
## S3 method for class 'pp'
summary(object, ...)
## S3 method for class 'wpp'
summary(object, ...)
```

### Arguments

| | |
|---|---|
| x, object | an object of class pgrid or pp or wpp. |
| ... | additional arguments. Currently without effect. |

### Details

Currently there is no difference between print and summary.

### Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>
Timo Wilm <timo.wilm@stud.uni-goettingen.de>

## pgrid *Constructor for the pgrid Class*

### Description

Construct an object of class "pgrid" from a matrix or a higher-dimensional array.

### Usage

```
pgrid(mass, boundary, gridtriple, generator, structure)
```

### Arguments

| | |
|---|---|
| mass | a matrix or higher-dimensional array specifing the masses in each pixel / at each pixel centre. |
| boundary, gridtriple, generator | arguments specifying the positions of the pixels. At most one of these can be specified. |
| structure | optional character string specifying the structure of the grid. Currently only "square" and "rectangular" make sense, and are derrived automatically from the dimensions of mass. |

### Details

For more detailed explanations of the arguments and other components of the derived object of class "pgrid", see pgrid-object.

### Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>

### See Also

Description of pgrid objects.

### Examples

```
m <- matrix(1:20, 4, 5)
a <- pgrid(m)
print(a)
print.default(a)

## Not run:
  plot(a, rot=TRUE)
## End(Not run)
```

---

## pgrid-object *Class of Pixel Grids*

---

### Description

The class "pgrid" (for pixel grid) represents regular quantizations of measures on (bounded subsets of) $R^d$. Currently only square quantizations of measures on a rectangles are supported, which in 2-d can be thought of as grey scale images.

### Details

Objects of class "pgrid" can be created by the function pgrid, and are most commonly used as input to the function transport. There are methods plot, print and summary for this class.

An object of class "pgrid" contains the following elements:

| | |
|---|---|
| structure | the structure of the grid. Currently only "square" and "rectangular" are supported. |
| dimension | the dimension $d$ of the space in which the grid is embedded. Must be $\geq 2$. |
| n | the number of pixels along the various coordinates, a vector of length dimension. |
| N | the total number of pixels. |
| boundary | the outer boundary of the "picture" (i.e. of the support of the measure). A vector of length 2*dimension, where the odd entries contain the left and the even entries contain the right endpoints of the various coordinates. |
| gridtriple | the rule for generating the pixel centres along the various coordinates. A dim by 3 matrix where each row is of the form c(start, end, step). |
| generators | the pixel centres along the various coordinates. A list of length dim where the i-th element is a vector of length n[i]. |
| mass | the array of masses in each pixel / at each pixel centre. In 2-d orientation corresponds to the standard orientation of images, see e.g. image. This means that pixels are arranged on coordinate axes in the order of their indices. |

## Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>

## See Also

Constructor function pgrid.

---

plot *Methods for Plotting Objects of Class pgrid, pp and wpp*

---

## Description

Methods for plotting objects of class pgrid, pp and wpp, possibly together with a transference plan.

## Usage

```
  ## S3 method for class 'pgrid'
plot(x, y = NULL, tplan = NULL, mass = c("colour", "thickness"),
   length = 0.1, acol, lwd, rot = FALSE, overlay = FALSE, static.mass =TRUE, ...)
  ## S3 method for class 'pp'
plot(x, y = NULL, tplan = NULL, cols = c(4, 2), cex = 0.8,
    acol = grey(0.3), lwd = 1, overlay = TRUE, ...)
  ## S3 method for class 'wpp'
plot(x, y = NULL, tplan = NULL, pmass=TRUE, tmass=TRUE, cols = c(4, 2),
    cex = 0.8, aglevel = 0.4, acol = grey(0.3), lwd = 1, overlay = TRUE, ...)
```

## Arguments

| | |
|---|---|
| x,y | one or two objects of class pgrid or class pp to be plotted. |
| tplan | a transference plan between the two objects x and y, typically an optimal transference plan obtained by a call to transport. |
| mass, pmass, tmass | for pgrid objects with a tplan: if mass == "colour", the mass transferred is depicted by heatmap colours; if mass == "thickness", it is depicted by the line widths of the arrows.<br>For wpp objects: pmass, tmass are logicals controlling whether the *amount* of mass associated with the points and the mass transferred should be depicted in the plot. |
| length | the length of the arrow heads in inches. |
| aglevel | for wpp objects with tmass = TRUE: the grey level chosen for depicting the transport of an average amount of mass. |
| acol | the colour of the arrows/lines of the transference plan. Ignored for pgrid objects if mass = "colour" and for wpp objects if tmass is TRUE. |
| cols | for pp objects: A vector of size 2 specifying the colours of the two pp objects. |
| cex,lwd,... | further graphic parameters used by plot. Note that for pgrid objects acol is ignored for mass == "colour", and lwd is ignored for mass == "thickness". Setting any of these parameters is optional. |
| rot | logical. Whether the mass matrices of pgrid objects should be rotated before calling image so that the orientation of the ploted pixelgrid and the orientation of the mass matrix are the same. Otherwise plotting follows the usual convention of image. |
| overlay | in the case of two objects x and y whether they should be plotted on top of one another (for pgrid objects the difference x-y is plotted) or not. In the presence of a transference plan overlay is forced to be true. |
| static.mass | for a transference plan that explicitly lists the "static mass transports" (i.e. mass that stays at the same site), should these transports also be plotted as disks with colours/sizes corresponding to the amount of mass that stays?<br>Note that it is wrong to assume that an optimal transference plan obtained by one of the algorithms will automatically list static mass transports. It is not the case for $p = 1$, where static mass transport at site $i$ is trivially equal to the minimum of source mass and target mass, and it is currently not the case for results obtained by method="aha". |

## Value

Used for its side effect.

## Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>

power_diagram *Compute the Power Diagram of Weighted Sites in 2-Dimensional Space*

## Description

Compute the power diagram of weighted sites in 2-dimensional space.

## Usage

```
power_diagram(xi, eta, w, rect = NA)
## S3 method for class 'power_diagram'
plot(x, weights=FALSE, add=FALSE, col=4, lwd=1, ...)
```

## Arguments

| | |
|---|---|
| xi,eta,w | vectors of equal length, where xi, eta are the coordinates of the sites and w are the corresponding weights. |
| rect | vetor of length 4. To get a finite representation of the power diagram, it will be intersected with the rectangle $[rect[1], rect[2]] \times [rect[3], rect[4]]$. Defaults to c(min(xi),max(xi),min(eta),max(eta)). |
| x | a power diagram as returned from power_diagram. |
| weights | logical. If TRUE, weights of non-redundant sites with non-negative weight are represented as circles whose radii are equal to the square roots of the corresponding weights. |
| add | logical. Should the power diagram be plotted on top of current graphics? |
| col,lwd,... | further arguments graphic parameters used by plot.default. |

## Details

The function power_diagram implements an algorithm by Edelsbrunner and Shah (1996) which computes regular triangulations and thus its dual representation, the power diagram. For point location, an algorithm devised by Devillers (2002) is used.

## Author(s)

Björn Bähre <bjobae@gmail.com>
(slightly modified by Dominic Schuhmacher <dschuhm1@uni-goettingen.de>)

## References

H. Edelsbrunner, N. R. Shah (1996), Incremental Topological Flipping Works for Regular Triangulations, Algorithmica 15, 223–241.

O. Devillers (2002), The Delaunay Hierarchy, International Journal of Foundations of Computer Science 13, 163–180.

## Examples

```
xi <- runif(100)
eta <- runif(100)
w <- runif(100,0,0.005)
x <- power_diagram(xi,eta,w,rect=c(0,1,0,1))
plot(x,weights=TRUE)
```

---

pp *Constructor for the pp Class*

---

## Description

Construct an object of class "pp" from a matrix.

## Usage

```
pp(coordinates)
```

## Arguments

| | |
|---|---|
| coordinates | a matrix specifying the coordinates of the points. Each row corresponds to a point. |

## Details

For more detailed explanations of the arguments and other components of the derived object of class "pp", see pp-object.

## Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>

## See Also

Description of pp objects.

## Examples

```
m <- matrix(c(1,1,2,2,3,1,4,2),4,2)
a <- pp(m)
print(a)
print.default(a)

## Not run:
  plot(a)
## End(Not run)
```

## pp-object *Class of (Unweighted) Point Patterns*

### Description

The class "pp" represents discrete measures with some fixed mass at any of finitely many locations.

### Details

Objects of class "pp" may be created by the function pp, and are most commonly used as input to the function transport. There are methods plot, print and summary for this class.

An object of class "pp" contains the following elements:

| | |
|---|---|
| dimension | the dimension of the Euclidean space in which the patterns live. Must be $\geq 2$. |
| N | the total number of points. |
| coordinates | the coordinates of the points. An N $\times$ dimension matrix, where each row corresponds to a point. |

### Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>

### See Also

Constructor function pp.

## random *Images to Illustrate the Use of transport.pgrid*

### Description

32 x 32, 64 x 64 and 128 x 128 images to illustrate the use of transport.pgrid. These are objects of class "pgrid".

### Usage

```
random32a

random32b

random64a

random64b

random128a

random128b
```

## Format

Objects of class 'pgrid'.

## Source

Randomly generated using the package `RandomFields`.

---

`semidiscrete` *Find Optimal Transport Partition Between pgrid and wpp.*

---

## Description

Given an object `a` of class `pgrid` specifying an image and an object `b` of class `wpp` specifiying a more flexible mass distribution at finitely many points, find the partition of the image (and hence the optimal transport map) that minimizes the total transport cost for going from `a` to `b`.

## Usage

```
semidiscrete(a, b, p = 2, method = c("aha"), control = list(), ...)
```

## Arguments

| | |
|---|---|
| `a` | an object of class `pgrid` usually representing an image or the discretization of a measure. |
| `b` | an object of class `wpp` usually having the same total mass as `a`. |
| `p` | the power $\geq 1$ to which the Euclidean distance between points is taken in order to compute costs. |
| `method` | the name of the algorithm to use. Currently only `aha` is supported. |
| `control` | a named list of parameters for the chosen method or the result of a call to `trcontrol`. Currently only the parameters `factr` and `maxit` can be set. |
| `...` | currently without effect. |

## Details

This is a wrapper function for the function `aha`. In the latter the Aurenhammer–Hoffmann–Aronov (1998) method is used with the multiscale approach presented in Mérigot (2011).

This function is not usually called by the user since it is automatically called by `transport` if the first argument is of class `pgrid` and the second argument is of class `wpp`.

## Value

An object of class `power_diagram` describing the optimal transport partition for `a` and `b`. Such an object has components `sites` and `cells`.

The former is a data.frame with columns `xi`, `eta` and `w` containing the $x$- and $y$-coordinates of the mass locations of `b` and the weight vector $w$ that describes the associated power tesselation uniquely. The latter is a list with as many components as mass locations of `b`, each giving a matrix containing in the columns the $x$- and $y$-coordinates of the polygonal cell associated with that location or `NULL` if the location has no cell.

Plotting methods exist both for objects of class `power_diagram` and for optimal transport maps represented by power diagrams.

## Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>
Björn Bähre <bjobae@gmail.com>

## References

F. Aurenhammer, F. Hoffmann and B. Aronov (1998). Minkowski-type theorems and least-squares clustering. Algorithmica 20(1), 61–76.

Q. Mérigot (2011). A multiscale approach to optimal transport. Computer Graphics Forum 30(5), 1583–1592. doi:10.1111/j.1467-8659.2011.02032.x

## See Also

`plot`, `transport`, `aha`

## Examples

```
##  See examples for function transport
```

---

`shielding` *Compute Optimal Transport (Cost/Plan) Using the Multiscale Shielding Method*

---

## Description

Runs the multiscale version of the Shielding Method (a.k.a. Short Cut Method) for computing the optimal transport (cost/plan) on a rectangular grid in $d$ dimensions for the squared Euclidean distance as cost function.

## Usage

```
shielding(a, b, nscales = 2, startscale = 1, flood = 0,
  measureScale = 1e-06, verbose = FALSE, basisKeep = 1, basisRefine = 1)
```

## Arguments

| | |
|---|---|
| a, b | arrays with $d$ coordinates representing source and target measure, respectively. The entries must be all positive. |
| nscales | the number of scales generated in the multiscale algorithm. |
| startscale | the first scale on which the problem is solved. |
| flood | a real number. If positive, take the maximum of entry and flood for each entry of a and b. |
| measureScale | the required precision for the entries. Computations are performed on round(a/measureScale) and the same for b using integer arithmetics. |
| verbose | logical. Toggles output to the console about the progress of the algorithm. |
| basisKeep, basisRefine | internal use only. |

## Details

If a and b do not have the same sum, they are normalized to sum 1 *before* flood and measureScale transformations are applied.

## Value

A list of components

| | |
|---|---|
| err | error code. 0 if everything is ok. |
| a_used,b_used | the vectorized arrays that were actually used by the algorithm. a, b after applying flood and measureScale. |
| coupling | a vectorized coupling describing the optimal transport from a_used to b_used |
| basis | a matrix with two columns describing the basis obtained for the optimal transport |
| u,v | vectors of optimal values in the dual problem |

## Use of CPLEX

For larger problems (thousands of grid points) there are considerable speed improvements when shielding can use the CPLEX numerical solver for the underlying constrained optimization problems. If a local installation of CPLEX is available, the transport package can be linked against it during installation. See the file src/Makevars in the source package for instructions.

## Author(s)

Bernhard Schmitzer <schmitzer@uni-muenster.de> and
Dominic Schuhmacher <dschuhm1@uni-goettingen.de>
(based on C++ code by Bernhard Schmitzer)

## References

B. Schmitzer (2016). A sparse multiscale algorithm for dense optimal transport. J. Math. Imaging Vision 56(2), 238–259. https://arxiv.org/abs/1510.05466

### See Also

transport, which calls this function if appropriate.

### Examples

```
## Not run:
shielding(random64a$mass,random64b$mass,nscales=5,measureScale=1)
## End(Not run)
```

---

`starting solutions` *Compute starting solution for the transportation problem*

---

### Description

Compute a feasible transference plan between two mass vectors.

### Usage

```
northwestcorner(a, b)
russell(a, b, costm)
```

### Arguments

| | |
|---|---|
| a,b | Two numeric vectors (typically containing natural numbers) of length $m$ and $n$, describing mass distributions. |
| costm | A $m$ by $n$ matrix of costs for moving one unit of mass. |

### Value

A list whose components are $m$ by $n$ matrices, viz.

| | |
|---|---|
| assignment | containing as $(i, j)$-th entry the mass assigned from origin $i$ to destination $j$; |
| basis | containing as $(i, j)$-th entry a 1 if it is a basic entry and a 0 otherwise. |

### Warnings

The current implementations are in R. Computations may be slow for larger vectors a and b.
The computed starting solution may be degenerate, i.e. there may be basic entries where zero mass is assigned.

### Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>

### See Also

transport

transport *Find Optimal Transport Plan Between Two Objects*

## Description

Given two objects a and b that specify distributions of mass and an object that specifies (a way to compute) costs, find the transport plan for going from a to b that minimizes the total cost.

## Usage

```
transport(a, b, ...)
## Default S3 method:
transport(a, b, costm, method = c("shortsimplex", "revsimplex", "primaldual"),
  control = list(), ...)
## S3 method for class 'pgrid'
transport(a, b, p = NULL, method = c("auto", "revsimplex", "shortsimplex",
  "shielding", "aha", "primaldual"), control = list(), ...)
## S3 method for class 'pp'
transport(a, b, p = 1, method = c("auction", "auctionbf", "shortsimplex",
  "revsimplex", "primaldual"), control = list(), ...)
## S3 method for class 'wpp'
transport(a, b, p = 1, method = c("revsimplex", "shortsimplex", "primaldual"),
  control = list(), ...)
```

## Arguments

| | |
|---|---|
| a, b | two objects that describe mass distributions, between which the optimal transport map is to be computed. For the default method these are vectors of non-negative values. For the other three methods these are objects of the respective classes. It is also possible to have a of class pgrid and b of class wpp. |
| costm | for the default method a length(a) by length(b) matrix specifying the cost of transporting single units of mass between the corresponding source and destination points. |
| p | for the three specialized methods the power $\geq 1$ to which the Euclidean distance between points is taken in order to compute costs. |
| method | the name of the algorithm to use. See details below. |
| control | a named list of parameters for the chosen method or the result of a call to trcontrol. Any parameters that are not set by the control argument will get reasonable (sometimes problem specific) defaults. |
| ... | currently without effect. |

## Details

There is a number of algorithms that are currently implemented and more will be added in future versions of the package. The following is a brief description of each key word used. Much more details can be found in the cited references and in a forthcoming package vignette.

aha: The Aurenhammer–Hoffmann–Aronov (1998) method with the multiscale approach presented in Mérigot (2011).

auction: The auction algorithm by Bertsekas (1988) with epsilon-scaling, see Bertsekas (1992).

auctionbf: A refined auction algorithm that combines forward and revers auction, see Bertsekas (1992).

primaldual: The primal-dual algorithm as described in Luenberger (2003, Section 5.9).

revsimplex: The revised simplex algorithm as described in Luenberger and Ye (2008, Section 6.4) with various speed improvements, including a multiscale approach.

shielding: The shielding (or shortcut) method, as described in Schmitzer (2016).

shortsimplex: The shortlist method based an a revised simplex algorithm, as described in Gottschlich and Schuhmacher (2014).

The order of the *default* key words specified for the argument method gives a rough idea of the relative efficiency of the algorithms for the corresponding class of objects. For a given a and b the actual computation times may deviate significantly from this order. For class pgrid the default method is "auto", which resolves to "revsimplex" if p is not 2 or the problem is very small, and to "shielding" otherwise.

The following table gives information about the applicability of the various algorithms (or sometimes rather their current implementations).

| | default | pgrid | pp | wpp |
|---|---|---|---|---|
| aha (p=2!) | - | + | - | @ |
| auction | - | - | + | - |
| auctionbf | - | - | + | - |
| primaldual | * | * | * | + |
| revsimplex | + | + | * | + |
| shielding (p=2!) | - | + | - | - |
| shortsimplex | + | + | * | + |

where: + recommended, * applicable (may be slow), - no implementation planned or combination does not make sense; @ indicates that the aha algorithm is available in the special combination where a is a pgrid object and b is a wpp object (and p is 2). For more details on this combination see the function semidiscrete.

Each algorithm has certain parameters supplied by the control argument. The following table gives an overview of parameter names and their applicability.

| | start | multiscale | individual parameters |
|---|---|---|---|
| aha ($p = 2!$) | - | + | factr, maxit |
| auction | - | - | lasteps, epsfac |
| auctionbf | - | - | lasteps, epsfac |
| primaldual | - | - | |
| revsimplex | + | + | |
| shielding ($p = 2!$) | - | + | |
| shortsimplex | - | - | slength, kfound, psearched |

start specifies the algorithm for computing a starting solution (if needed). Currently the Modified Row Minimum Rule (start="modrowmin"), the North-West Corner Rule (start="nwcorner") and the method by Russell (1969) (start="russell") are implemented. When start="auto" (the default) the ModRowMin Rule is chosen. However, for transport.pgrid and p larger than 1, there are two cases where an automatic multiscale procedure is also performed, i.e. the optimal transport is first computed on coarser grids and information from these solutions is then used for the finer girds. This happens for method = "revsimplex", where a single coarsening at factor scmult=2 is performed, and for method = "shielding", where a number of coarsenings adapted to the dimensions of the array is performed.

For p=1 and method="revsimplex", as well as p=2 and method="aha" there are multiscale versions of the corresponding algorithms that allows for finer control via the parameters nscales, scmult and returncoarse. The default value of nscales=1 suppresses the multiscale version. For larger problems it is advisable to use the multiscale version, which currently is only implemented for square pgrids in two dimensions. The algorithm proceeds then by coarsening the pgrid nscales-1 times, summarizing each time scmult^2 pixels into one larger pixels, and then solving the various transport problems starting from the coarsest and using each previous problem to compute a starting solution to the next finer problem. If returncoarse is TRUE, the coarser problems and their solutions are returned as well (revsimplex only).

factr, maxit are the corresponding components of the control argument in the optim L-BFGS-B method.

lasteps, epsfac are parameters used for epsilon scaling in the auction algortihm. The algorithm starts with a "transaction cost" per bid of epsfac^k * lasteps for some reasonable k generating finer and finer approximate solutions as the k counts down to zero. Note that in order for the procedure to make sense, epsfac should be larger than one (typically two- to three-digit) and in order for the final solution to be exact lasteps should be smaller than 1/n, where n is the total number of points in either of the point patterns. slength, kfound, psearched are the shortlist length, the number of pivot candidates needed, and the percentage of shortlists searched, respectively.

## Value

A data frame with columns from, to and mass that specifies from which element of a to which element of b what amount of mass is sent in the optimal transport plan. For class pgrid elements are specified as vector indices in terms of the usual column major enumeration of the matrices a$mass and b$mass. There are plot methods for the classes pgrid and pp, which can plot this solution.

If returncoarse is TRUE for the revsimplex method, a list with components sol and prob giving the solutions and problems on the various scales considered. The solution on the finest scale (i.e. the output we obtain when setting returncoarse to FALSE) is in sol[[1]].

If a is of class pgrid and b of class wpp (and p=2), an object of class power_diagram as described in the help for the function semidiscrete. The plot method for class pgrid can plot this solution.

## Use of CPLEX

The combination of the shielding-method with the CPLEX numerical solver outperforms the other algorithms by an order of magnitude for large problems (only applicable for p=2 and objects of class "pgrid"). If a local installation of CPLEX is available, the transport package can be linked against it during installation. See the file src/Makevars in the source package for instructions.

## Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>
Björn Bähre <bjobae@gmail.com>
(code for aha-method)
Carsten Gottschlich <gottschlich@math.uni-goettingen.de>
(original java code for shortlist- and revsimplex-methods)
Bernhard Schmitzer <schmitzer@uni-muenster.de>
(code for shielding-method)

## References

F. Aurenhammer, F. Hoffmann and B. Aronov (1998). Minkowski-type theorems and least-squares clustering. Algorithmica 20(1), 61–76.

D. P. Bertsekas (1988). The auction algorithm: a distributed relaxation method for the assignment problem. Annals of Operations Research 14(1), 105–123.

D. P. Bertsekas (1992). Auction algorithms for network flow problems: a tutorial introduction. Computational Optimization and Applications 1, 7–66.

C. Gottschlich and D. Schuhmacher (2014). The shortlist method for fast computation of the earth mover's distance and finding optimal solutions to transportation problems. PLOS ONE 9(10), e110214. doi:10.1371/journal.pone.0110214

D.G. Luenberger (2003). Linear and nonlinear programming, 2nd ed. Kluwer.

D.G. Luenberger and Y. Ye (2008). Linear and nonlinear programming, 3rd ed. Springer.

Q. Mérigot (2011). A multiscale approach to optimal transport. Computer Graphics Forum 30(5), 1583–1592. doi:10.1111/j.1467-8659.2011.02032.x

B. Schmitzer (2016). A sparse multiscale algorithm for dense optimal transport. J. Math. Imaging Vision 56(2), 238–259. https://arxiv.org/abs/1510.05466

## See Also

plot, wasserstein

## Examples

```
#
# example for the default method
#
a <- c(100, 200, 80, 150, 50, 140, 170, 30, 10, 70)
b <- c(60, 120, 150, 110, 40, 90, 160, 120, 70, 80)
set.seed(24)
costm <- matrix(sample(1:20, 100, replace=TRUE), 10, 10)
res <- transport(a,b,costm)

# pretty-print solution in matrix form for very small problems:
transp <- matrix(0,10,10)
transp[cbind(res$from,res$to)] <- res$mass
rownames(transp) <- paste(ifelse(nchar(a)==2," ",""),a,sep="")
colnames(transp) <- paste(ifelse(nchar(b)==2," ",""),b,sep="")
print(transp)
```

```
#
# example for class 'pgrid'
#
dev.new(width=9, height=4.5)
par(mfrow=c(1,2), mai=rep(0.1,4))
image(random32a$mass, col = grey(0:200/200), axes=FALSE)
image(random32b$mass, col = grey(0:200/200), axes=FALSE)
res <- transport(random32a,random32b)
dev.new()
par(mai=rep(0,4))
plot(random32a,random32b,res,lwd=1)


#
# example for class 'pp'
#
set.seed(27)
x <- pp(matrix(runif(400),200,2))
y <- pp(matrix(runif(400),200,2))
res <- transport(x,y)
dev.new()
par(mai=rep(0.02,4))
plot(x,y,res)


#
# example for class 'wpp'
#
set.seed(30)
m <- 30
n <- 60
massx <- rexp(m)
massx <- massx/sum(massx)
massy <- rexp(n)
massy <- massy/sum(massy)
x <- wpp(matrix(runif(2*m),m,2),massx)
y <- wpp(matrix(runif(2*n),n,2),massy)
res <- transport(x,y,method="revsimplex")
plot(x,y,res)


#
# example for semidiscrete transport between class 'pgrid' and class 'wpp'
#
set.seed(33)
n <- 100
massb <- rexp(n)
massb <- massb/sum(massb)*1e5
b <- wpp(matrix(runif(2*n),n,2),massb)
res <- transport(random32a,b,p=2)
plot(random32a,b,res)
```

transport_track *Create a Dynamic Visualization of a Transference Plan Between Two pgrids*

## Description

Given two objects `source` and `target` of class `pgrid` and a transference plan, typically the result of a call to `transport`, create an animation of the dynamic transference plan (a.k.a. displacement interpolation)

## Usage

```
transport_track(source, target, tplan, K = 50, scmult = 1, smooth = FALSE,
   H = matrix(c(1,0,0,1),2,2), create.file = c("none","gif_im"),
   file.name = "Rtransport.gif", delay.time = 20, cut = FALSE)
```

## Arguments

| | |
|---|---|
| `source, target` | objects of class `pgrid`. |
| `tplan` | a transference plan between `source` and `target`, typically an optimal transference plan obtained by a call to `transport`. |
| `K` | the number of intermediate frames to be produced between `source` and `target`. |
| `scmult` | the factor by which the number of pixels in each dimension is multiplied to obtain a smoother rendering of the dynamic transference plan. |
| `smooth` | logical. Whether a kernel smoothing or a linear binning procedure is used to generate the images. Defaults to `FALSE`. |
| `H` | the bandwith matrix used to perform the two dimensional kernel density estimation or the linear binning respectively. |
| `create.file` | the file type to be created or `"none"` to return only an array of intermediate mass distributions. |
| `file.name` | the path for the output file. Ignored if `create.file` is `"none"`. |
| `delay.time` | the delay time introduced in ticks. One tick is usually 1/100 of a second, but may depend on individual options set in ImageMagick. |
| `cut` | logical. Whether the boundary pixels are cut off. Currently the only way to deal with the edge effect (see Details). |

## Details

The intermediate frames are produced by the interpolation formula $[(1-t)\mathrm{pr}_1 + t\mathrm{pr}_2]_{\#}\pi$, where $\pi$ is the transference plan, $\mathrm{pr}_1$ and $\mathrm{pr}_2$ are the first and second coordinate projections of $\mathbf{R}^2 \times \mathbf{R}^2$ onto $\mathbf{R}^2$, and $t \in \{0, 1/(K+1), \ldots, K/(K+1), 1\}$. If $\pi$ is an optimal transference plan this yields the displacement interpolation, at least if we assume as underlying cost function the Euclidean metric to the $p$-th power, where $p = 1, 2$.

The kernel smoothing procedure gives usually nicer animations, but takes several orders of magnitudes longer.

There are currently visible edge effects in both the kernel smoothing and the linear binning procedure that lead to darker pixels at the boundary of the image. The cut parameter may be used to remove the boundary pixels completely and thus produce a smaller output. The edge will be dealt with more adequatly in future versions.

Conversion to an animated gif is performed by a system call to the convert tool of ImageMagick. The latter may have to be installed first.

## Value

An array containing the various interpolation images.

Unless `create.file="none"`, the function is mainly used for its side effect (saving a file to the specified path). So the array is returned invisibly.

## Warning

Running this function with `smooth=TRUE` and even moderate `K` can take a long time!

## Author(s)

Florian Heinemann <florian.heinemann@stud.uni-goettingen.de>
(slightly modified by Dominic Schuhmacher <dschuhm1@uni-goettingen.de>)

## See Also

`Function` `transport` for computing optimal transference plans.

## Examples

```
tplan <- transport(random32a,random32b)
series <- transport_track(random32a, random32b, tplan, scmult=3, create.file="none")
dev.new(width=16,height=8)
oldpar <- par(mfrow=c(5,10), mai=rep(0.01,4))
for (i in 1:50) {
  image(series[,,i], col=grey(seq(0,1,0.005)), asp=1, axes=FALSE,zlim=c(min(series),max(series)))
}
par(oldpar)
```

---

# trcontrol *Set the Control Parameters Used by* `transport`.

---

## Description

Set the control parameters for the algorithm used by the function `transport`.

## Usage

```
trcontrol(method = c("revsimplex", "shortsimplex", "primaldual", "aha",
          "shielding", "auction", "auctionbf"), para = list(), start = c("auto",
          "modrowmin", "nwcorner", "russell"), nscales = 1, scmult = 2,
          returncoarse = FALSE, a = NULL, b = NULL, M = NULL, N = NULL)
```

## Arguments

| | |
|---|---|
| `method` | The algorithm to be used to compute the optimal transference plan. See details for the function `transport.pgrid`. |
| `para` | A list of parameters that are specific to the chosen method. See the table on the help page of the function `transport`. |
| `start` | If `method == "revsimplex"`, the method for computing a starting solution. |
| `nscales, scmult, returncoarse` | The parameters for the multiscale versions of certain algorithms. See the help on `transport`. |
| `a,b,M,N` | The two objects `a` and `b` for which the transportation problem is to be solved *or* the sizes `M` and `N` of these objects. Based on the information available here, `trcontrol` tries hard to find reasonable values for the control parameters of the algorithm not specified directly. |

## Details

For further details about the parameters of the individual algorithms see the help page for `transport`.

## Value

A list with components `method`, `para`, `start`, `nscales`, `scmult`, `returncoarse` as entered or adapted/computed based on the arguments `method`, `a`, `b`, `M`, `N`.

## Note

This function is typically only called by the user to check what the parameter settings used by the function `transport` are for a given problem.

## Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>

## See Also

`transport`

`wasserstein` *Compute the Wasserstein Distance Between Two Objects*

## Description

Given two objects `a` and `b` that specify measures in $R^d$, compute the Wasserstein distance of order `p` between the objects.

## Usage

```
wasserstein(a, b, p=1, tplan=NULL, prob=TRUE, ...)
```

## Arguments

| | |
|---|---|
| `a, b` | two objects that describe mass distributions in $R^d$. Either both of class `pgrid` or `pp` or `wpp`. Their dimension $d$ must be at least 2; see function `wasserstein1d` for $d = 1$. |
| `p` | the power $\geq 1$ to which the Euclidean distance between points is taken in order to compute transportation costs. |
| `tplan` | an optional transference plan in the format returned by the function `transport`. If `NULL` an optimal transference plan based on `a`, `b` and `p` is computed by a call to `transport`. |
| `prob` | logical. Should the objects `a`, `b` be interpreted as probability measures, i.e. their total mass be normalized to 1? |
| `...` | further parameters passed to `transport` if `tplan` is `NULL`. |

## Details

The Wasserstein distance of order `p` is defined as the `p`-th root of the total cost incurred when transporting measure `a` to measure `b` in an optimal way, where the cost of transporting a unit of mass from $x$ to $y$ is given as the `p`-th power $\|x - y\|^p$ of the Euclidean distance.

If `tplan` is supplied by the user, no checks are performed whether it is optimal for the given problem. So this function may be used to compare different (maybe suboptimal) transference plans with regard to their total costs.

For further details on the algorithms used, see help of `transport`.

## Value

A single number, the Wasserstein distance for the specified problem.

## Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>

## See Also

plot, transport, wasserstein1d

## Examples

```
#
# example for class 'pgrid'
#
wasserstein(random32a,random32b,p=1)
res <- transport(random32a,random32b,p=2)
wasserstein(random32a,random32b,p=1,res)
# is larger than above:
# the optimal transport for p=2 is not optimal for p=1

#
# example for class 'pp'
#
set.seed(27)
x <- pp(matrix(runif(500),250,2))
y <- pp(matrix(runif(500),250,2))
wasserstein(x,y,p=1)
wasserstein(x,y,p=2)
```

---

# wasserstein1d *Compute the Wasserstein Distance Between Two Univariate Samples*

---

## Description

Given two vectors a and b, compute the Wasserstein distance of order p between their empirical distributions.

## Usage

```
wasserstein1d(a, b, p = 1, wa = NULL, wb = NULL)
```

## Arguments

| | |
|---|---|
| a, b | two vectors. |
| p | a positive number. The order of the Wasserstein distance. |
| wa, wb | optional vectors of weights for a and b. |

## Details

The Wasserstein distance of order p is defined as the p-th root of the total cost incurred when transporting a pile of mass into another pile of mass in an optimal way, where the cost of transporting a unit of mass from $x$ to $y$ is given as the p-th power $\|x - y\|^p$ of the Euclidean distance.

In the present function the vector a represents the locations on the real line of $m$ deposits of mass $1/m$ and the vector b the locations of $n$ deposits of mass $1/n$. If the user specifies weights wa and wb, these default masses are replaced by wa/sum(wa) and wb/sum(wb), respectively.

In terms of the empirical distribution function $F(t) = \sum_{i=1}^{m} w_i^{(a)} 1\{a_i \leq t\}$ of locations $a_i$ with normalized weights $w_i^{(a)}$, and the corresponding function $G(t) = \sum_{j=1}^{n} w_j^{(b)} 1\{b_j \leq t\}$ for b, the Wasserstein distance in 1-d is given as

$$W_p(F,G) = \left( \int_0^1 |F^{-1}(u) - G^{-1}(u)|^p \, du \right)^{1/p},$$

where $F^{-1}$ and $G^{-1}$ are generalized inverses. If $p = 1$, we also have

$$W_1(F,G) = \int_{-\infty}^{\infty} |F(x) - G(x)| \, dx.$$

## Value

A single number, the Wasserstein distance for the specified data.

## Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>

## See Also

wasserstein

## Examples

```
x <- rnorm(200)
y <- rnorm(150,2)
wasserstein1d(x,y)
```

---

# wpp *Constructor for the wpp Class*

---

## Description

Construct an object of class "wpp" from a matrix of points and a vector of masses.

## Usage

```
wpp(coordinates, mass)
```

## Arguments

| | |
|---|---|
| coordinates | a matrix specifying the coordinates of the points. Each row corresponds to a point. |
| mass | a vector of non-negative values specifying the masses at these points. |

## Details

For more detailed explanations of the arguments and other components of the returned object of class "wpp", see wpp-object.

It is legitimate to assign mass 0 to individual points in the arguments. However, when constructing the wpp-object such points are deleted. The coordinates of the deleted points can still be accessed via the attribute zeropoints.

## Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>
Timo Wilm <timo.wilm@stud.uni-goettingen.de>

## See Also

Description of pp objects.

## Examples

```
m <- matrix(c(1,1,2,2,3,1,4,2),4,2)
a <- pp(m)
print(a)
print.default(a)

## Not run:
  plot(a)
## End(Not run)
```

---

# wpp-object *Class of Weighted Point Patterns*

---

## Description

The class "wpp" represents discrete measures with positive mass at any of finitely many locations.

## Details

Objects of class "wpp" may be created by the function wpp, and are most commonly used as input to the function transport. There are methods plot, print and summary for this class.

An object of class "wpp" contains the following elements:

| | |
|---|---|
| dimension | the dimension of the Euclidean space in which the patterns live. Must be $\geq 2$. |
| N | the total number of point. |
| coordinates | the coordinates of the points. An N $\times$ dimension matrix, where each row corresponds to a point. |
| mass | the masses at these points. A vector of length N of positive numbers. |
| totmass | the total mass of the point pattern. |

## Author(s)

Dominic Schuhmacher <dschuhm1@uni-goettingen.de>
Timo Wilm <timo.wilm@stud.uni-goettingen.de>

## See Also

Constructor function wpp.

# Index