

Package ‘seriation’

January 2, 2012

Type Package

Title Infrastructure for seriation

Author Michael Hahsler, Christian Buchta and Kurt Hornik

Maintainer Michael Hahsler <michael@hahsler.net>

Version 1.0-6

Date 2011-10-18

Description Infrastructure for seriation with an implementation of several seriation/sequencing techniques to reorder matrices, dissimilarity matrices, and dendrograms. Also contains some visualizations techniques based on seriation.

Classification/ACM G.1.6, G.2.1, G.4

URL <http://R-Forge.R-project.org/projects/seriation/>,
<http://lyle.smu.edu/IDA/seriation/>

Depends R (>= 2.10.0), stats, cluster, TSP, gclus, grid, colorspace

Suggests MASS, biclust

License GPL-2

Copyright The code in src/bea.f is Copyright (C) 1991 F. Murtagh; src/bbwrcg.f, src/arsa.f and src/bburcg.f are Copyright (C) 2005 M. Brusco, H.F. Kohn, and S. Stahl. All other code is Copyright (C) Michael Hahsler, Christian Buchta, and Kurt Hornik.

Repository CRAN

Date/Publication 2011-10-19 09:34:18

R topics documented:

bertinplot	2
criterion	4
criterion_methods	8
dissplot	9
get_order	12
hmap	13
Irish	15
Munsingen	16
permute	17
pimage	18
Psych24	20
seriate	21
seriation_methods	25
ser_permutation	27
ser_permutation_vector	28
Townships	29
Zoo	29
Index	31

bertinplot	<i>Plot a Bertin matrix</i>
------------	-----------------------------

Description

Plot a data matrix of cases and variables. Each value is represented by a symbol. Large values are highlighted. The matrix can be rearranged to make structure in the data visible (see Falguerolles et al 1997). `bertin_cut_line()` can be used to add cut lines (see Details).

Usage

```
bertinplot(x, order = NULL, highlight = TRUE, options = NULL)
```

Arguments

x	a data matrix. Note that following Bertin, columns are variables and rows are cases. This behavior can be reversed using <code>reverse = TRUE</code> in options.
order	an object of class <code>ser_permutation</code> to rearrange x before plotting. If <code>NULL</code> , no rearrangement is performed.
highlight	a logical scalar indicating whether to use highlighting. If <code>TRUE</code> , all variables with values greater than the variable-wise mean are highlighted. To control highlighting, also a logical matrix with the same dimensions as x can be supplied.
options	a list with options for plotting. The list can contain the following elements:

`panel.function` a function to produce the symbols. Currently available functions are `panel.bars` (default), `panel.circles`, `panel.squares`, `panel.blocks` and `panel.lines`. For circles and squares neg. values are represented by a dashed border. For blocks all blocks are the same size (can be used with `shading=TRUE`).

`reverse` logical indicating whether to swap cases and variables in the plot. The default (FALSE) is to plot cases as columns and variables as rows.

`xlab`, `ylab` labels (default: use labels from `x`).

`spacing` relative space between symbols (default: 0.2).

`shading` use gray shades to encode value instead of highlighting (default: FALSE).

`frame` plot a grid to separate symbols (default: codeFALSE).

`mar` margins (see `par`).

`gp_labels` `gpar` object for labels (see `gpar`).

`gp_panels` `gpar` object for panels (see `gpar`).

`newpage` a logical indicating whether to start the plot on a new page (see `grid.newpage`).

`pop` a logical indicating whether to pop the created viewports (see `pop.viewport`)?

Details

The plot is organized as a matrix of symbols. The symbols are drawn by a `panel` function, where all symbols of a row are drawn by one call of the function (using vectorization). The interface for the `panel` function is `panel.myfunction(value, spacing, hl)`. `value` is the vector of values for a row scaled between 0 and 1, `spacing` contains the relative space between symbols and `hl` is a logical vector indicating which symbol should be highlighted.

Cut lines can be added to an existing bertin plot using `bertin_cut_line(x=NULL, y=NULL)`. `x/y` is can be a number indicating where to draw the cut line between two columns/rows. If both `x` and `y` is specified then one can select a row/column and the other can select a range to draw a line which does only span a part of the row/column. It is important to call `bertinplot()` with the option `pop=FALSE`.

References

de Falguerolles, A., Friedrich, F., Sawitzki, G. (1997): A Tribute to J. Bertin's Graphical Data Analysis. In: Proceedings of the SoftStat '97 (Advances in Statistical Software 6), 11–20.

See Also

[ser_permutation](#), [seriate](#), Package [grid](#).

Examples

```
data("Irish")
scale_by_rank <- function(x) apply(x, 2, rank)
x <- scale_by_rank(Irish[,-6])

## use the the sum of absolute rank differences
order <- c(
  seriate(dist(x, "minkowski", p = 1)),
```

```

  seriate(dist(t(x), "minkowski", p = 1))
)

## plot
bertinplot(x, order)

## some alternative displays
bertinplot(x, order, options = list(shading = TRUE, panel = panel.blocks))
bertinplot(x, order, options = list(panel = panel.lines))
bertinplot(x, order, options = list(panel = panel.squares))
bertinplot(x, order,
  options = list(panel = panel.circles, spacing = -0.5))

## plot with cut lines (we manually set the order here)
order <- ser_permutation(c(21, 16, 19, 18, 14, 12, 20, 15,
  17, 26, 13, 41, 7, 11, 5, 23, 28, 34, 31, 1, 38, 40,
  3, 39, 4, 27, 24, 8, 37, 36, 25, 30, 33, 35, 2,
  22, 32, 29, 10, 6, 9),
  c(4, 2, 1, 6, 8, 7, 5, 3))

bertinplot(x, order, options=list(pop=FALSE))
bertin_cut_line(,4) ## horizontal line between rows 4 and 5
bertin_cut_line(,7) ## separate "Right to Life" from the rest
bertin_cut_line(14,c(0,4)) ## separate a block of large values (vertically)

```

criterion

Criterion for a loss/merit function for data given a permutation

Description

Compute the value for different loss functions L and merit function M for data given a permutation.

Usage

```
criterion(x, order = NULL, method = NULL)
```

Arguments

x	an object of class <code>dist</code> or a matrix (currently no functions are implemented for array).
order	an object of class <code>ser_permutation</code> suitable for x. If <code>NULL</code> , the identity permutation is used.
method	a character vector with the names of the criteria to be employed, or <code>NULL</code> (default) in which case all available criteria are used.

Details

For a symmetric dissimilarity matrix D with elements $d(i, j)$ where $i, j = 1 \dots p$, the aim is generally to place low distance values close to the diagonal. The following criteria to judge the quality of a certain permutation of the objects in a dissimilarity matrix are currently implemented:

"Gradient_raw", "Gradient_weighted" Gradient measures (Hubert et al 2001).

A symmetric dissimilarity matrix where the values in all rows and columns only increase when moving away from the main diagonal is called a perfect *anti-Robinson matrix* (Robinson 1951). A suitable merit measure which quantifies the divergence of a matrix from the anti-Robinson form is

$$M(D) = \sum_{i < k < j} f(d_{ij}, d_{ik}) + \sum_{i < k < j} f(d_{ij}, d_{kj})$$

where $f(\cdot, \cdot)$ is a function which defines how a violation or satisfaction of a gradient condition for an object triple (O_i, O_k, O_j) is counted.

Hubert et al (2001) suggest two functions. The first function is given by:

$$f(z, y) = \text{sign}(y - z) = +1 \quad \text{if } z < y; \quad 0 \quad \text{if } z = y; \quad \text{and} \quad -1 \quad \text{if } z > y.$$

It results in raw number of triples satisfying the gradient constraints minus triples which violate the constraints.

The second function is defined as:

$$f(z, y) = |y - z| \text{sign}(y - z) = y - z$$

It weights the each satisfaction or violation by the difference by its magnitude given by the absolute difference between the values.

"AR_events", "AR_deviations" Anti-Robinson events (Chen 2002). An even simpler loss function can be created in the same way as the gradient measures above by concentrating on violations only.

$$L(D) = \sum_{i < k < j} f(d_{ik}, d_{ij}) + \sum_{i < k < j} f(d_{kj}, d_{ij})$$

To only count the violations we use

$$f(z, y) = I(z, y) = 1 \quad \text{if } z < y \quad \text{and} \quad 0 \quad \text{otherwise.}$$

$I(\cdot)$ is an indicator function returning 1 only for violations. Chen (2002) presented a formulation for an equivalent loss function and called the violations *anti-Robinson events* and also introduced a weighted versions of the loss function resulting in

$$f(z, y) = |y - z| I(z, y)$$

using the absolute deviations as weights.

"Path_length" Hamiltonian path length (Caraux and Pinloche 2005).

The order of the objects in a dissimilarity matrix corresponds to a path through a graph where each node represents an object and is visited exactly once, i.e., a Hamilton path. The length of the path is defined as the sum of the edge weights, i.e., dissimilarities.

$$L(D) = \sum_{i=1}^{n-1} d_{i,i+1}$$

The length of the Hamiltonian path is equal to the value of the minimal span loss function (as used by Chen 2002). Both notions are related to the *traveling salesperson problem (TSP)*.

If order is not unique or there are non-finite distance values NA is returned.

"Inertia" Inertia criterion (Caraux and Pinloche 2005).

Measures the moment of the inertia of dissimilarity values around the diagonal as

$$M(D) = \sum_i \sum_j d(i,j) |i-j|^2.$$

$|i-j|$ is used as a measure for the distance to the diagonal and $d(i,j)$ gives the weight. This criterion gives higher weight to values farther away from the diagonal. It increases with quality.

"Least_squares" Least squares criterion (Caraux and Pinloche 2005).

The sum of squares of deviations between the dissimilarities and rank differences (in the matrix) between two elements:

$$L(D) = \sum_i \sum_j (d(i,j) - |i-j|)^2,$$

where $d(i,j)$ is an element of the dissimilarity matrix D and $|i-j|$ is the rank difference between the objects.

Note that if Euclidean distance is used to calculate D from a data matrix X , the order of the elements in X by projecting them on the first principal component of X minimizes this criterion. The least squares criterion is related to *unidimensional scaling*.

For a general matrix $X = x_{ij}$, $i = 1 \dots m$ and $j = 1 \dots n$, currently the following loss/merit functions are implemented:

"ME" Measure of Effectiveness (McCormick 1972).

The measure of effectiveness (ME) for matrix X , is defined as

$$M(X) = 1/2 \sum_{i=1}^n \sum_{j=1}^m x_{i,j} (x_{i,j-1} + x_{i,j+1} + x_{i-1,j} + x_{i+1,j})$$

with, by convention

$$x_{0,j} = x_{m+1,j} = x_{i,0} = x_{i,n+1} = 0.$$

ME is a merit measure, i.e. a higher ME indicates a better arrangement. Maximizing ME is the objective of the bond energy algorithm (BEA).

"Moore_stress", "Neumann_stress" Stress (Niermann 2005).

Stress measures the conciseness of the presentation of a matrix/table and can be seen as a purity function which compares the values in a matrix/table with its neighbors. The stress measure used here is computed as the sum of squared distances of each matrix entry from its adjacent entries. The following types of neighborhoods are available:

Moore: comprises the eight adjacent entries.

Neumann: comprises the four adjacent entries.

The major difference between the Moore and the Neumann neighborhood is that for the later the contribution of row and column permutations to stress are independent and thus can be optimized independently.

Value

A named vector of real values.

References

G. Caraux and S. Pinloche (2005): Permutmatrix: A Graphical Environment to Arrange Gene Expression Profiles in Optimal Linear Order, *Bioinformatics*, **21**(7), 1280–1281.

C.-H. Chen (2002): Generalized association plots: Information visualization via iteratively generated correlation matrices, *Statistica Sinica*, **12**(1), 7–29.

L. Hubert, P. Arabie, and J. Meulman (2001): *Combinatorial Data Analysis: Optimization by Dynamic Programming*. Society for Industrial Mathematics.

S. Niermann (2005): Optimizing the Ordering of Tables With Evolutionary Computation, *The American Statistician*, **59**(1), 41–46.

W.S. Robinson (1951): A method for chronologically ordering archaeological deposits, *American Antiquity*, **16**, 293–301.

W.T. McCormick, P.J. Schweitzer and T.W. White (1972): Problem decomposition and data reorganization by a clustering technique, *Operations Research*, **20**(5), 993-1009.

Examples

```
## create random data and calculate distances
m <- matrix(runif(10),ncol=2)
d <- dist(m)

## get an order for rows (optimal for the least squares criterion)
o <- seriate(m, method = "PCA", margin = 1)
o

## compare the values for all available criteria
rbind(
  unordered = criterion(d),
  ordered = criterion(d, o)
)
```

criterion_methods *Registry for criterion methods*

Description

A registry to manage methods to calculate a criterion value given data and a permutation.

Usage

```
list_criterion_methods(kind)
show_criterion_methods(kind)
get_criterion_method(kind, name)
set_criterion_method(kind, name, definition, description = NULL, merit = NA, ...)
```

Arguments

kind	the data type the method works on. For example, "dist", "matrix" or "array".
name	a short name for the method used to refer to the method in the function <code>criterion()</code> .
definition	a function containing the method's code.
description	a description of the method. For example, a long name.
merit	a boolean indicating if the criterion measure is a merit (TRUE) or a loss (FALSE) measure.
...	further information that is stored for the method in the registry.

Details

`list_criterion_method()` lists all available methods for a given data type (`kind`). The result is a vector of character strings with the short names of the methods.

`show_criterion_method()` shows all available methods for a given data type (`kind`) including a description.

`get_criterion_method()` returns information (including the implementing function) about a given method in form of an object of class "criterion_method".

With `set_criterion_method()` new criterion methods can be added by the user. The implementing function (`definition`) needs to have the formal arguments `x`, `order`, `...`, where `x` is the data object, `order` is an object of class `permutation_vector` and `...` can contain additional information for the method passed on from `criterion()`. The implementation has to return the criterion value as a scalar.

Examples

```
list_criterion_methods("dist")
show_criterion_methods("dist")
get_criterion_method("dist", "AR_d")
```

```

## define a new method

## a function that return sum of the diagonal elements
criterion_method_matrix_foo <- function(x, order, ...) {
  if(!is.null(order)) x <- permute(x,order)
  sum(diag(x))
}

## set new method
set_criterion_method("matrix", "foo", criterion_method_matrix_foo,
  "foo: a useless demo criterion", FALSE)

list_criterion_methods("matrix")

##use all criterion methods (including the new one)
criterion(matrix(1:9, ncol=3))

```

dissplot

Dissimilarity Plot

Description

Visualizes a dissimilarity matrix using seriation and matrix shading. Entries with lower dissimilarities (higher similarity) are plotted darker. Such a plot can be used to uncover hidden structure in the data.

The plot can also be used to visualize cluster quality (see Ling 1973). Objects belonging to the same cluster are displayed in consecutive order. The placement of clusters and the within cluster order is obtained by a seriation algorithm which tries to place large similarities/small dissimilarities close to the diagonal. Compact clusters are visible as dark squares (low dissimilarity) on the diagonal of the plot. Additionally, a Silhouette plot (Rousseeuw 1987) is added. This visualization is similar to CLUSION (see Strehl and Ghosh 2002), however, allows for using arbitrary seriating algorithms.

Usage

```
dissplot(x, labels = NULL, method = NULL, control = NULL, options = NULL)
```

Arguments

x	an object of class <code>dist</code> .
labels	NULL or an integer vector of the same length as rows/columns in <code>x</code> indicating the cluster membership for each object in <code>x</code> as consecutive integers starting with one. The labels are used to reorder the matrix.
method	a list with up to three elements or a single character string. Use a single character string to apply the same algorithm to reorder the clusters (inter cluster seriation) as well as the objects within each cluster (intra cluster seriation). If separate algorithms for inter and intra cluster seriation are required, <code>method</code> can be a list of two named elements (<code>inter_cluster</code> and <code>intra_cluster</code>)

each containing the name of the respective seriation method. See `seriate.dist` for available algorithms.

Set method to NA to plot the matrix as is (no or only coarse seriation). For intra cluster reordering the special method `silhouette width` is available. Objects in clusters are then ordered by silhouette width (from silhouette plots). If no method is given, the default method of `seriate.dist` is used.

The third list element (named `aggregation`) controls how inter cluster dissimilarities are computed from the given dissimilarity matrix. The choices are "avg" (average pairwise dissimilarities; average-link), "min" (minimal pairwise dissimilarities; single-link), "max" (maximal pairwise dissimilarities; complete-link), and "Hausdorff" (pairs up each point from one cluster with the most similar point from the other cluster and then uses the largest dissimilarity of paired up points).

control	a list of control options passed on to the seriation algorithm. In case of two different seriation algorithms, control can contain a list of two named elements (<code>inter_cluster</code> and <code>intra_cluster</code>) containing each a list with the control options for the respective algorithm.
options	<p>a list with options for plotting the matrix. The list can contain the following elements:</p> <p>plot a logical indicating if a plot should be produced. if FALSE, the returned object can be plotted later using the function <code>plot</code> which takes as the second argument a list of plotting options (see options below).</p> <p><code>cluster_labels</code> a logical indicating whether to display cluster labels in the plot.</p> <p><code>averages</code> a logical vector of length two. The first element controls the upper triangle and the second element the lower triangle of the plot. FALSE displays the original dissimilarity between objects, TRUE displays cluster-wise average dissimilarities, and NA leaves the triangle white (default: <code>c(FALSE, TRUE)</code>), i.e., the lower triangle displays averages)</p> <p><code>lines</code> a logical indicating whether to draw lines to separate clusters.</p> <p><code>flip</code> a logical indicating if the clusters are displayed on the diagonal from north-west to south-east (FALSE; default) or from north-east to south-west (TRUE).</p> <p><code>silhouettes</code> a logical indicating whether to include a silhouette plot (see Rousseeuw, 1987).</p> <p><code>threshold</code> a numeric. If used, only plot distances below the threshold are displayed.</p> <p><code>col</code> colors used for the image plot. If <code>col</code> is a single number, it specifies the number of different shades used in the plot (default: 100 shades of gray using the HCL colorspace). If <code>col</code> not a single number, it is expected to be a full color palette and the options <code>hue</code> and <code>power</code> are ignored.</p> <p><code>hue</code> color in [0,360] taken from the HCL color wheel. NA indicates gray scale (default: gray scale).</p> <p><code>power</code> determines how luminance (and chroma for color) should be increased (1 = linear, 2 = quadratic, etc.) with dissimilarity.</p> <p><code>colorkey</code> a logical indicating whether to place a color key below the plot.</p>

main title for the plot.
 lines_col color used for the lines to separate clusters.
 newpage a logical indicating whether to start plot on a new page (see `grid.newpage` in package **grid**).
 pop a logical indicating whether to pop the created viewports (see package **grid**)?
 gp an object of class `gpar` containing graphical parameters (see `gpar` in package **grid**).

Value

An invisible object of class `cluster_proximity_matrix` with the following elements:

order	NULL or integer vector giving the order used to plot x.
cluster_order	NULL or integer vector giving the order of the clusters as plotted.
method	vector of character strings indicating the seriation methods used for plotting x.
k	NULL or integer scalar giving the number of clusters generated.
description	a <code>data.frame</code> containing information (label, size, average intra-cluster dissimilarity and the average silhouette) for the clusters as displayed in the plot (from top/left to bottom/right).

This object can be used for plotting via `plot(x, options = NULL, ...)`, where `x` is the object and `options` contains a list with plotting options (see above).

References

- Ling, R.F. (1973): A computer generated aid for cluster analysis. *Communications of the ACM*, **16**(6), 355–361.
- Rousseeuw, P.J. (1987): Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, **20**(1), 53–65.
- Strehl, A. and Ghosh, J. (2003): Relationship-based clustering and visualization for high-dimensional data mining. *INFORMS Journal on Computing*, **15**(2), 208–230.

See Also

`dist` (in package **stats**), package **grid** and `seriate`.

Examples

```

data("iris")
d <- dist(iris[-5])

## plot original matrix
res <- dissplot(d, method = NA)

## plot reordered matrix using the nearest insertion algorithm (from tsp)
res <- dissplot(d, method = "tsp",
  options = list(main = "Seriation (TSP)"))

```

```

## cluster with pam (we know iris has 3 clusters)
library("cluster")
l <- pam(d, 3, cluster.only = TRUE)

## we use a grid layout to place several plots on a page
grid.newpage()
pushViewport(viewport(layout=grid.layout(nrow = 2, ncol = 2),
  gp = gpar(fontsize = 8)))
pushViewport(viewport(layout.pos.row = 1, layout.pos.col = 1))

## visualize the clustering
res <- disspplot(d, l, method = "chen",
  options = list(main = "PAM + Seriation (Chen) - standard",
    newpage = FALSE))

popViewport()
pushViewport(viewport(layout.pos.row = 1, layout.pos.col = 2))

## more visualization options
## color: use 10 shades of blue (hue = 270)
plot(res, options = list(main = "PAM + Seriation (Chen) - blue, only avg.",
  col = 10, hue = 260, averages = c(TRUE, TRUE), newpage = FALSE))

popViewport()
pushViewport(viewport(layout.pos.row = 2, layout.pos.col = 1))

## threshold and cubic scale to highlight differences
plot(res, options = list(main = "PAM + Seriation (Chen) - threshold",
  threshold = 1.5, power = 3, newpage = FALSE))

popViewport()
pushViewport(viewport(layout.pos.row = 2, layout.pos.col = 2))

## use custom (logistic) scale
plot(res, options = list(main = "PAM + Seriation (Chen) - logistic scale",
  col = hcl(c = 0, l = (plogis(seq(0,10,length=100),
  location = 2, scale = 1/2, log = FALSE))*100),
  newpage = FALSE))

popViewport(2)

## the reordered_cluster_dissimilarity_matrix object
res
names(res)

```

Description

Method to get an integer permutation vector from an object of class `ser_permutation` or `ser_permutation_vector`.

Usage

```
get_order(x, ...)

## S3 method for class 'ser_permutation_vector'
get_order(x, ...)
## S3 method for class 'ser_permutation'
get_order(x, dim = 1, ...)
```

Arguments

<code>x</code>	an object of class <code>ser_permutation</code> or <code>ser_permutation_vector</code>
<code>dim</code>	which dimension should be returned?
<code>...</code>	further arguments (unused).

Value

Returns an integer vector.

See Also

[ser_permutation_vector](#), [ser_permutation](#)

Examples

```
## permutation_vector
o <- ser_permutation_vector(1:10)
o

get_order(o)

## permutation
o2 <- ser_permutation(o, 5:1)
o2

get_order(o2, 2)
```

Description

Provides several reordered versions of heat map including dendrogram based reordering with optimal leaf order and matrix seriation based heat maps.

Usage

```
hmap(x, distfun = dist, hclustfun = hclust,
     method = NULL, control = NULL, options = NULL, ...)
```

Arguments

x	a matrix.
distfun	function used to compute the distance (dissimilarity) between both rows and columns (default: dist).
hclustfun	function used for hierarchical clustering. If hclustfun = NULL, no hierarchical clustering is performed and matrix based seriation is performed to reorder rows and columns for the heat map.
method	a character strings indicating the used seriation algorithm (see <code>seriate.dist</code> for dendrogram seriation and for matrix based seriation).
control	a list of control options passed on to the seriation algorithm.
options	a list with arguments for plotting. The following arguments are possible: main title for the plot. col colors used for the plot (default: gray colors). For matrix based seriation (hclustfun = NULL), the following additional arguments are possible: gp an object of class <code>gpar</code> containing graphical parameters (see <code>gpar</code> in package grid). newpage a logical indicating whether to start plot on a new page (see <code>gpar</code> in package grid). prop a logical indicating whether the height and width of x should be plotted proportional to its dimensions.
...	further arguments. For dendrogram based heat maps the arguments are passed on to <code>heatmap</code> in stats (Note that the following arguments cannot be used: <code>Rowv</code> , <code>Colv</code> , <code>distfun</code> , <code>hclustfun</code> , <code>reorderfun</code> , <code>scale</code>). For dendrogram = FALSE further arguments are used as options (see above).

Details

For dendrogram = TRUE, `seriate.hclust` with the default method "optimal" is used for arranging the dendrograms and `x.heatmap` is used for plotting.

For dendrogram = FALSE, `seriate.dist` with the default method "tsp" (a traveling salesperson solver) for arranging x is used. **grid** code implemented in this package is used to produce the plot.

Note that unlike the default behavior of `heatmap`, scaling is not automatically applied. The data have to be scaled before using `hmap`.

Value

An invisible list with elements:

`rowInd`, `colInd` index permutation vectors.

`reorder_method` name of the method used to reorder the matrix.

For `dendrogram = TRUE` the list can contain additional elements (see `heatmap` for details).

See Also

[seriate.dist](#) and [heatmap](#) in **stats**.

Examples

```
data("Zoo")
x <- as.matrix(Zoo[, -17])
x <- scale(x, center = FALSE)

## optimally reordered heatmap
hmap(x)

## heatmap with seriated distance matrices
hmap(x, hclustfun = NULL)

## with proportional display
hmap(x, hclustfun = NULL, options = list(prop = TRUE,
    main = "Zoo Data"))
```

Irish

Irish data

Description

A data matrix containing the results of 8 referenda for 41 Irish communities (two values are missing) used in Falguerolles et al (1997).

Usage

```
data(Irish)
```

Format

The format is a 41 x 9 matrix.

Details

Column 6 contains the size of the Electorate in 1992.

Source

The data was kindly provided by Guenter Sawitzki.

References

de Falguerolles, A., Friedrich, F., Sawitzki, G. (1997): A Tribute to J. Bertin's Graphical Data Analysis. In: Proceedings of the SoftStat '97 (Advances in Statistical Software 6), 11–20.

Examples

```
data(Irish)
```

Munsingen

Hodson's Munsingen Data Set

Description

This data set contains a grave times artifact incidence matrix for the Celtic Münsingen-Rain cemetery in Switzerland as provided by Hodson (1968) and published by Kendall 1971.

Usage

```
data("Munsingen")
```

Format

A 59 x 70 0-1 matrix. Rows (graves) and columns (artifacts) are in the order determined by Hodson (1968).

References

Hodson, F.R. (1968): *The La Tene Cemetery at Münsingen-Rain*. Stämpfli, Bern.

Kendall, D.G. (1971): Seriation from abundance matrices. In: Hodson, F.R., Kendall, D.G. and Tautu, P., (Editors). *Mathematics in the Archaeological and Historical Sciences*, Edinburgh University Press, Edinburgh, 215–232.

Examples

```
data("Munsingen")

## Seriation method after Kendall (1971)
## Kendall's square symmetric matrix S and SoS
S <- function(x, w = 1) {
  sij <- function(i, j) w * sum(pmin(x[i,], x[j,]))
  h <- nrow(x)
  r <- matrix(ncol = h, nrow = h)
  for(i in 1:h) for (j in 1:h) r[i,j] <- sij(i,j)
  r
}

SoS <- function(x) S(S(x))
```

```

## Kendall's horse shoe (Hamiltonian arc)
horse_shoe_plot <- function(mds, sigma, threshold = mean(sigma), ...) {
  plot(mds, main = paste("Kendall's horse shoe with th =", threshold), ...)
  l <- which(sigma > threshold, arr.ind=TRUE)
  for(i in 1:nrow(l)) lines(rbind(mds[l[i,1],], mds[l[i,2],]))
}

## shuffle data
x <- Munsingen[sample(nrow(Munsingen)),]

## calculate matrix and do isoMDS (from package MASS)
sigma <- SoS(x)
library("MASS")
mds <- isoMDS(1/(1+sigma))$points

## plot Kendall's horse shoe
horse_shoe_plot(mds, sigma)

## find order using a TSP
tour <- solve_TSP(insert_dummy(TSP(dist(mds)), label = "cut"),
  method = "2-opt", control = list(rep = 15))
tour <- cut_tour(tour, "cut")
lines(mds[tour,], col = "red", lwd = 2)

## create and plot order
order <- ser_permutation(tour, 1:ncol(x))
bertinplot(x, order, options= list(panel=panel.circles,
  rev = TRUE))

## compare criterion values
rbind(
  random = criterion(x),
  reordered = criterion(x, order),
  Hodson = criterion(Munsingen)
)

```

permute

Permute a dist object, a matrix, an array, a list, or a numeric vector

Description

A method for permuting

various classes including the observations in a `dist` object, the rows and columns of a matrix, and all dimensions of an array given a suitable `ser_permutation` object.

Usage

```
permute(x, order)
```

Arguments

x	an object (a list, a numeric vector, a dist object, a matrix, an array or any other object which provides dim and standard subsetting with "[").
order	an object of class ser_permutation which contains suitable permutation vectors for x.

Details

The permutation vectors in ser_permutation are suitable if the number of permutation vectors matches the number of dimensions of x and if the length of each permutation vector has the same length as the corresponding dimension of x.

For 1-dimensional/1-mode data (list, vector, dist), order can also be a single permutation vector of class ser_permutation_vector or data which can be automatically coerced to this class (e.g. a numeric vector).

See Also

[ser_permutation](#), [dist](#) in package **stats**.

Examples

```
m <- matrix(rnorm(10), 5, 2)
m

permute(m, ser_permutation(5:1, 2:1))

d <- dist(m)
d

permute(d, ser_permutation(c(3,2,1,4,5)))

## this also works for 1-mode data:
# permute(d, ser_permutation_vector(c(3,2,1,4,5)))
# permute(d, c(3,2,1,4,5))
```

pimage

Permutation image plot

Description

Provides methods for plotting image plots for matrix and dist objects given a permutation.

Usage

```
pimage(x, order = NULL, col = NULL, main = "", xlab = "", ylab = "", axes = TRUE, ..., newpage=TRUE, pop=TR

## S3 method for class 'matrix'
pimage(x, order=NULL, col=NULL, main="", xlab="", ylab="",
       axes=TRUE, ..., newpage=TRUE, pop=TRUE)
## S3 method for class 'dist'
pimage(x, order = NULL, col = NULL, main="", xlab="", ylab="",
       axes = TRUE, upper.tri = TRUE, lower.tri = TRUE, ...,
       newpage=TRUE, pop=TRUE)
```

Arguments

x	a matrix or an object of class <code>dist</code> .
order	an object of class <code>ser_permutation</code> . If <code>NULL</code> the order in <code>x</code> is plotted.
col	a list of colors used. If <code>NULL</code> , a gray scale is used (for matrix larger values are displayed darker and for <code>dist</code> smaller distances are darker). For matrices containing logical data, black and white is used.
main	Plot title.
xlab, ylab	labels for the x and y axes.
axes	a logical indicating whether to add axes using the labels of <code>x</code> . The default value (<code>TRUE</code>) shows only axes if the dimension is below 10 (so it is reasonable to show labels).
upper.tri, lower.tri	a logical indicating whether to show the upper or lower triangle of the distance matrix.
...	further arguments passed on to <code>image</code> in graphics .
newpage, pop	two logical. Start plot on a new page and pop the viewports after plotting (see <code>Grid</code>).

Details

Plots a matrix in its original row and column orientation. This means, in a plot the columns become the x-coordinates and the reversed rows the y-coordinates.

If `x` is of class `dist` it is converted to full-storage representation before plotting.

Author(s)

Christian Buchta and Michael Hahsler

Examples

```
x <- matrix(sample(c(FALSE, TRUE), 150, rep=TRUE), ncol=10)

## matrix
pimage(x, main = "random data")
```

```
## plot seriated matrix
pimage(x, seriate(x), col = c("white", "green"), main = "reordered data")

## distances
d <- dist(x, method = "binary")
pimage(d, lower.tri = FALSE)

pimage(d, seriate(d), lower.tri = FALSE)
```

 Psych24

Psych24 – 24 psychological tests

Description

A data set collected by Holzinger and Swineford (1939) which consists of the results of 24 psychological tests given to 145 seventh and eighth grade students in a Chicago suburb. This data set contains the correlation matrix for the 24 test results.

The data set was also used as an example for visualization of cluster analysis by Ling (1973).

Usage

```
data("Psych24")
```

Format

A 24 x 24 correlation matrix.

References

Holzinger, K. L., Swineford, F. (1939): A study in factor analysis: The stability of a bi-factor solution. *Supplementary Educational Monograph*, No. **48**. Chicago: University of Chicago Press.

Ling, R. L. (1973): A computer generated aid for cluster analysis. *Communications of the ACM*, **16**(6), pp. 355–361.

Examples

```
data("Psych24")

## create a dist object and also get rid of the one negative entry in the
## correlation matrix
d <- as.dist(1 - abs(Psych24))

pimage(d)

## do hclust as in Ling (1973)
hc <- hclust(d, method = "complete")
plot(hc)

pimage(d, hc)
```

```
## use seriation
order <- seriate(d, method = "tsp")
#order <- seriate(d, method = "tsp", control = list(method = "concorde"))
pimage(d, order)
```

seriate

Seriate objects in dissimilarity matrices, matrices or arrays

Description

Tries to find an linear order for objects using data in form of a dissimilarity matrix (two-way one mode data), a data matrix (two-way two-mode data) or a data array (k-way k-mode data).

Usage

```
## S3 method for class 'dist'
seriate(x, method = NULL, control = NULL, ...)
## S3 method for class 'matrix'
seriate(x, method = NULL, control = NULL,
        margin = c(1,2), ...)
## S3 method for class 'array'
seriate(x, method = NULL, control = NULL,
        margin = seq(length(dim(x))), ...)
```

Arguments

x	the data.
method	a character string with the name of the seriation method (default: varies by data type).
control	a list of control options passed on to the seriation algorithm.
margin	a vector giving the margins to be seriated. For matrix, 1 indicates rows, 2 indicates columns, c(1,2) indicates rows and columns. For array, margin gets a vector with the dimensions to seriate.
...	further arguments (unused).

Details

Two-way two-mode data has to be provided as a dist object (not as a symmetric matrix). Similarities have to be transformed in a suitable way into dissimilarities. Currently the following methods are implemented for dist:

"ARSA" Anti-Robinson seriation by simulated annealing.

A heuristic developed by Brusco et al (2007).

"BBURCG" Anti-Robinson seriation (unweighted)

A branch-and-bound implementation by Brusco and Stahl (2005).

"BBWRG" Anti-Robinson seriation (weighted)

A branch-and-bound implementation by Brusco and Stahl (2005).

"TSP" Traveling salesperson problem solver.

A solver in **TSP** can be used (see `solve_TSP` in package **TSP**). The solver method can be passed on via the `control` argument, e.g. `control = list(method = "insertion")`.

Since a tour returned by a TSP solver is a connected circle and we are looking for a path representing a linear order, we need to find the best cutting point. Climer and Zhang (2006) suggest to add a dummy city with equal distance to each other city before generating the tour. The place of this dummy city in an optimal tour with minimal length is the best cutting point (it lies between the most distant cities).

"Chen" Rank-two ellipse seriation (Chen 2002).

This method starts with generating a sequence of correlation matrices R^1, R^2, \dots, R^1 is the correlation matrix of the original distance matrix D (supplied to the function as x), and

$$R^{n+1} = \phi R^n,$$

where ϕ calculates the correlation matrix.

The rank of the matrix R^n falls with increasing n . The first R^n in the sequence which has a rank of 2 is found. Projecting all points in this matrix on the first two eigenvectors, all points fall on an ellipse. The order of the points on this ellipse is the resulting order.

The ellipse can be cut at the two interception points (top or bottom) of the vertical axis with the ellipse. In this implementation the top most cutting point is used.

"MDS" Multidimensional scaling (MDS).

Use multidimensional scaling techniques to find an linear order. Note that unidimensional scaling would be more appropriate but is very hard to solve. Generally, MDS provides good results.

By default, metric MDS (`cmdscale` in **stats**) is used. In case of of general dissimilarities, non-metric MDS can be used. The choices are `isoMDS` and `sammon` from **MASS**. The method can be specified as the element `method` ("`cmdscale`", "`isoMDS`" or "`sammon`") in `control`.

"HC" Hierarchical clustering.

Using the order of the leaf nodes in a dendrogram obtained by hierarchical clustering can be used as a very simple seriation technique. This method applies hierarchical clustering (`hclust`) to x . The clustering method can be given using a `"method"` element in the `control` list. If omitted, the default `"complete"` is used.

"GW", "OLO" Hierarchical clustering with optional reordering.

Uses also the order of the leaf nodes in a dendrogram (see method "HC"), however, the leaf notes are reordered.

A dendrogram (binary tree) has 2^{n-1} internal nodes (subtrees) and the same number of leaf orderings. That is, at each internal node the left and right subtree (or leaves) can be swapped, or, in terms of a dendrogram, be flipped.

Method "GW" uses an algorithm developed by Gruvaeus and Wainer (1972) and implemented in package **gclus** (Hurley 2004). The clusters are ordered at each level so that the objects at the edge of each cluster are adjacent to that object outside the cluster to which it is nearest. The method produces an unique order.

Method "OLO" (Optimal leaf ordering, Bar-Joseph et al., 2001) produces an optimal leaf ordering with respect to the minimizing the sum of the distances along the (Hamiltonian) path

connecting the leaves in the given order. The time complexity of the algorithm is $O(n^3)$. Note that non-finite distance values are not allowed.

Both methods start with a dendrogram created by `hclust`. As the "method" element in the control list a clustering method (default "complete") can be specified. Alternatively, a `hclust` object can be supplied using an element named "hclust".

Two-way two mode data are general positive matrices. Currently the following methods are implemented for matrix:

"BEA" Bond Energy Algorithm (BEA; McCormick 1972).

The algorithm tries to maximize the measure of effectiveness (see `criterion`) of a non-negative matrix. Due to the definition of this measure, the tasks of ordering rows and columns is separable and can be solved independently.

A row is arbitrarily placed; then rows are positioned one by one. When this is completed, the columns are treated similarly. The overall procedure amounts to two approximate traveling salesperson problems (TSP), one on the rows and one on the columns. The so-called 'best insertion strategy' is used: rows (or columns) are inserted into the current permuted list of rows (or columns). Several consecutive runs of the algorithm might improve the energy.

Note that Arabie and Hubert (1990) question its use with non-binary data if the objective is to find a seriation or one-dimensional orderings of rows and columns.

The BEA code used in this package was implemented by Fionn Murtagh.

In control as element "rep" the number of runs can be specified. The results of the best run will be returned.

"BEA_TSP" Use a TSP to optimize the measure of effectiveness (Lenstra 1974).

Use a TSP solver to optimize ME.

In control as element "method" a TSP solver method can be specified (see package **TSP**).

"PCA" Principal component analysis.

Uses the projection of the data on its first principal component to determine the order.

Note that for a distance matrix calculated from `x` with Euclidean distance, this methods minimizes the least square criterion.

For array no built-in methods are currently available.

Value

Returns an object of class `ser_permutation`.

References

- P. Arabie and L.J. Hubert (1990): The bond energy algorithm revisited, *IEEE Transactions on Systems, Man, and Cybernetics*, **20**(1), 268–274.
- Z. Bar-Joseph, E. D. Demaine, D. K. Gifford, and T. Jaakkola. (2001): Fast Optimal Leaf Ordering for Hierarchical Clustering. *Bioinformatics*, **17**(1), 22–29.
- Brusco, M., Koehn, H.F., and Stahl, S. (2007): Heuristic Implementation of Dynamic Programming for Matrix Permutation Problems in Combinatorial Data Analysis. *Psychometrika*, conditionally accepted.

- Brusco, M., and Stahl, S. (2005): *Branch-and-Bound Applications in Combinatorial Data Analysis*. New York: Springer.
- Chen, C. H. (2002): Generalized Association Plots: Information Visualization via Iteratively Generated Correlation Matrices. *Statistica Sinica*, **12**(1), 7–29.
- Sharlee Climer, Weixiong Zhang (2006): Rearrangement Clustering: Pitfalls, Remedies, and Applications, *Journal of Machine Learning Research*, **7**(Jun), 919–943.
- Gruvaeus, G. and Wainer, H. (1972): Two Additions to Hierarchical Cluster Analysis, *British Journal of Mathematical and Statistical Psychology*, **25**, 200–206.
- Hurley, Catherine B. (2004): Clustering Visualizations of Multidimensional Data. *Journal of Computational and Graphical Statistics*, **13**(4), 788–806.
- J.K. Lenstra (1974): Clustering a Data Array and the Traveling-Salesman Problem, *Operations Research*, **22**(2) 413–414.
- W.T. McCormick, P.J. Schweitzer and T.W. White (1972): Problem decomposition and data reorganization by a clustering technique, *Operations Research*, **20**(5), 993–1009.

See Also

[criterion](#), [solve_TSP](#) in **TSP**, [hclust](#) in **stats**

Examples

```
##seriate dist
data("iris")
x <- as.matrix(iris[-5])
x <- x[sample(1:nrow(x)),]
d <- dist(x)

## default seriation
order <- seriate(d)
order

## plot
def.par <- par(no.readonly = TRUE)
layout(cbind(1,2), respect = TRUE)

pimage(d, main = "Random")
pimage(d, order, main = "Reordered")

par(def.par)

## compare quality
rbind(
  random = criterion(d),
  reordered = criterion(d, order)
)

## seriate matrix
data("iris")
```

```

x <- as.matrix(iris[-5])

## to make the variables comparable, we scale the data
x <- scale(x, center = FALSE)

## try some methods
def.par <- par(no.readonly = TRUE)
layout(matrix(1:4, ncol = 2, byrow = TRUE), respect=TRUE)

pimage(x, main = "original data")
criterion(x)

order <- seriate(x, method = "BEA_TSP")
pimage(x, order, main = "TSP to optimize ME")
criterion(x, order)

order <- seriate(x, method="PCA")
pimage(x, order, main = "first principal component")
criterion(x, order)

## 2 TSPs
order <- c(
  seriate(dist(x), method = "TSP"),
  seriate(dist(t(x)), method = "TSP")
)
pimage(x, order, main = "2 TSPs")
criterion(x, order)

par(def.par)

```

seriation_methods *Registry for seriation methods*

Description

A registry to manage methods for seriation.

Usage

```

list_seriation_methods(kind)
show_seriation_methods(kind)
get_seriation_method(kind, name)
set_seriation_method(kind, name, definition, description = NULL, ...)

```

Arguments

kind	the data type the method works on. For example, "dist", "matrix" or "array".
name	a short name for the method used to refer to the method in seriate().
definition	a function containing the method's code.

description a description of the method. For example, a long name.
 ... further information that is stored for the method in the registry.

Details

`list_seriation_method()` lists all available methods for a given data type (kind). The result is a vector of character strings with the short names of the methods.

`show_seriation_method()` shows all available methods including a description.

`get_seriation_method()` returns information (including the implementing function) about a given method in form of an object of class "seriation_method".

With `set_seriation_method()` new seriation methods can be added by the user. The implementing function (definition) needs to have the formal arguments `x`, `control`, where `x` is the data object and `control` contains a list with additional information for the method passed on from `seriate()`. The implementation has to return a list of objects which can be coerced into `ser_permutation_vector` objects (e.g., integer vectors). The elements in the list have to be in corresponding order to the dimensions of `x`.

Examples

```
show_seriation_methods("matrix")

list_seriation_methods("matrix")

get_seriation_method("matrix", "BEA")

## define a new method

## create a identity function which returns the identity order
seriation_method_identity <- function(x, control) {
  lapply(dim(x), seq)
}

## set new method
set_seriation_method("matrix", "identity", seriation_method_identity,
  "Identity order")

set_seriation_method("array", "identity", seriation_method_identity,
  "Identity order")

show_seriation_methods("matrix")

##use all criterion methods (including the new one)
seriate(matrix(1:12, ncol=3), "identity")
```

ser_permutation	<i>Class ser\permutation – A collection of permutation vectors for seri- ation</i>
-----------------	--

Description

The class `ser_permutation` is a collection of permutation vectors (see class `ser_permutation_vector`), one for each dimension (mode) of the data to be permuted.

Usage

```
## constructor  
ser_permutation(x, ...)
```

Arguments

<code>x</code>	an object of class <code>ser_permutation_vector</code> or any object which can be converted into a object of class <code>ser_permutation</code> (e.g. an integer vector).
<code>...</code>	permutation vectors for further dimensions

See Also

[ser_permutation_vector](#)

Examples

```
o <- ser_permutation(1:5, 10:1)  
o  
  
## length (number of dimensions)  
length(o)  
  
## get permutation vector for 2nd dimension  
get_order(o, 2)  
  
## reverse dimensions  
o[2:1]  
  
## combine  
o <- c(o, ser_permutation(1:15))  
o  
  
## get an individual permutation  
o[[2]]
```

ser_permutation_vector

Class ser_permutation_vector – A single permutation vector for seriation

Description

The class ser_permutation_vector represents a single permutation vector.

Usage

```
## constructor
ser_permutation_vector(x, method = NULL)
```

Arguments

x	an object which contains a permutation vector (currently an integer vector or an object of class hclust)
method	a string representing the method used to obtain the permutation vector

Details

ser_permutation_vector objects are usually packed into a ser_permutation object which is a collection of k permutation vectors for k -mode data.

The constructor ser_permutation_vector checks if the permutation vector is valid (i.e. if all integers occur exactly once).

See Also

[ser_permutation](#)

Examples

```
p <- ser_permutation_vector(1:10, "identity")
p

## some methods
length(p)
get_order(p)
get_method(p)
```

Townships

Bertin's Characteristics and Townships Data Set

Description

This data set was used to illustrate that the conciseness of presentation can be improved by seriating the rows and columns.

Usage

```
data("Townships")
```

Format

A matrix with 16 0-1 variables (columns) indicating the presence (1) or absence (0) of characteristics of townships (rows).

References

Bertin, J. (1981): *Graphics and Graphic Information Processing*. Berlin, Walter de Gruyter.

Examples

```
data("Townships")

## original data
pimage(Townships)
criterion(Townships)

## seriated data
order <- seriate(Townships, method = "BEA", control = list(rep = 5))
pimage(Townships, order)
criterion(Townships, order)
```

Zoo

Zoo database

Description

A database containing characteristics of different animals. The database was created and donated by Richard S. Forsyth.

Usage

```
data("Zoo")
```

Format

A data frame with 101 observations on the following 17 variables.

```
hair {0, 1}
feathers {0, 1}
eggs {0, 1}
milk {0, 1}
airborne {0, 1}
aquatic {0, 1}
predator {0, 1}
toothed {0, 1}
backbone {0, 1}
breathes {0, 1}
venomous {0, 1}
fins {0, 1}
legs Numeric (set of values: {0, 2, 4, 5, 6, 8})
tail {0, 1}
domestic {0, 1}
catsize {0, 1}
class a factor with levels amphibian bird fish insect invertebrate mammal reptile
```

Source

D.J. Newman, S. Hettich, C.L. Blake and C.J. Merz (1998): UCI Repository of machine learning databases, <http://www.ics.uci.edu/~mllearn/MLRepository.html>, University of California, Irvine, Dept. of Information and Computer Sciences.

Examples

```
data("Zoo")
x <- scale(Zoo[, -17])

d <- dist(x)
pimage(d)

order <- seriate(d, method = "tsp")
pimage(d, order)
```

Index

- *Topic **classes**
 - ser_permutation, 27
 - ser_permutation_vector, 28
- *Topic **cluster**
 - bertinplot, 2
 - criterion, 4
 - dissplot, 9
 - seriate, 21
- *Topic **datasets**
 - Irish, 15
 - Munsingen, 16
 - Psych24, 20
 - Townships, 29
 - Zoo, 29
- *Topic **hplot**
 - bertinplot, 2
 - dissplot, 9
 - hmap, 13
 - pimage, 18
- *Topic **manip**
 - get_order, 12
 - permute, 17
- *Topic **misc**
 - criterion_methods, 8
 - seriation_methods, 25
- *Topic **optimize**
 - seriate, 21
- bertin_cut_line (bertinplot), 2
- bertinplot, 2
- c.ser_permutation (ser_permutation), 27
- criterion, 4, 24
- criterion_methods, 8
- dissplot, 9
- dist, 11, 18
- get_criterion_method (criterion_methods), 8
- get_method (ser_permutation_vector), 28
- get_order, 12
- get_seriation_method (seriation_methods), 25
- hclust, 24
- heatmap, 15
- hmap, 13
- Irish, 15
- length.ser_permutation_vector (ser_permutation_vector), 28
- list_criterion_methods (criterion_methods), 8
- list_seriation_methods (seriation_methods), 25
- Munsingen, 16
- panel.bars (bertinplot), 2
- panel.blocks (bertinplot), 2
- panel.circles (bertinplot), 2
- panel.lines (bertinplot), 2
- panel.squares (bertinplot), 2
- permutation (ser_permutation), 27
- permutation_vector (ser_permutation_vector), 28
- permute, 17
- pimage, 18
- plot.reordered_cluster_dissimilarity_matrix (dissplot), 9
- print.reordered_cluster_dissimilarity_matrix (dissplot), 9
- print.ser_permutation (ser_permutation), 27
- print.ser_permutation_vector (ser_permutation_vector), 28
- Psych24, 20
- ser_permutation, 3, 13, 18, 27, 28

ser_permutation_vector, [13](#), [27](#), [28](#)
seriate, [3](#), [11](#), [21](#)
seriate.dist, [15](#)
seriation_methods, [25](#)
set_criterion_method
 (criterion_methods), [8](#)
set_seriation_method
 (seriation_methods), [25](#)
show_criterion_methods
 (criterion_methods), [8](#)
show_seriation_methods
 (seriation_methods), [25](#)
solve_TSP, [24](#)

Townships, [29](#)

Zoo, [29](#)