Package ‘blockmodeling’

February 15, 2020

Type Package
Title Generalized and Classical Blockmodeling of Valued Networks
Version 0.3.6
Date 2020-1-28
Imports stats, methods, Matrix, parallel
Suggests sna, doRNG, doParallel, foreach
Author Aleš Žiberna [aut, cre]
Maintainer Aleš Žiberna <ales.ziberna@gmail.com>

Description
This is primarily meant as an implementation of generalized blockmodeling for valued networks.
In addition, measures of similarity or dissimilarity based on structural equivalence and
regular equivalence (REGE algorithms) can be computed and partitioned matrices can be plotted:

License GPL (>= 2)
Encoding UTF-8
RoxygenNote 6.0.1
NeedsCompilation yes
Repository CRAN
Date/Publication 2020-02-15 21:20:16 UTC

R topics documented:

blockmodeling-package .................................................. 2
clu ................................................................. 3
critFunC ............................................................ 5
find.m .............................................................. 11
formatA ............................................................. 13
fun.by.blocks ......................................................... 14
genMatrixMult ......................................................... 15
genRandomPar ......................................................... 16
An R package for Generalized and classical blockmodeling of valued networks

Description

This package is primarily meant as an implementation of Generalized blockmodeling. In addition, functions for computation of (dis)similarities in terms of structural and regular equivalence, plotting and other "utility" functions are provided.

Author(s)

Aleš Žiberna

References


See Also

Packages: sna network

Functions inside this package: critFunC, optParC, optRandomParC, REGE, plot.mat

Examples

n <- 8 # If larger, the number of partitions increases dramatically,
# as does if we increase the number of clusters
net <- matrix(NA, ncol = n, nrow = n)
clu <- rep(1:2, times = c(3, 5))
tclu <- table(clu)
net[clu == 1, clu == 1] <- rnorm(n = tclu[1] * tclu[1], mean = 0, sd = 1)
net[clu == 1, clu == 2] <- rnorm(n = tclu[1] * tclu[2], mean = 4, sd = 1)
net[clu == 2, clu == 1] <- rnorm(n = tclu[2] * tclu[1], mean = 0, sd = 1)
net[clu == 2, clu == 2] <- rnorm(n = tclu[2] * tclu[2], mean = 0, sd = 1)

# We select a random partition and then optimize it
all.par <- nkpartitions(n = n, k = length(tclu))
# Forming the partitions
all.par <- lapply(apply(all.par, 1, list), function(x) x[[1]])

# Optimizing one partition
res <- optParC(M = net,
               clu = all.par[[sample(1:length(all.par), size = 1)]]
               approaches = "hom", homFun = "ss", blocks = "com")
plot(res) # Hopefully we get the original partition

# Optimizing 10 random partitions with optRandomParC
res <- optRandomParC(M = net, k = 2, rep = 10,
                     approaches = "hom", homFun = "ss", blocks = "com")
plot(res) # Hopefully we get the original partition

# Using indirect approach - structural equivalence
D <- sedist(M = net)
plot.mat(net, clu = cutree(hclust(d = D, method = "ward"), k = 2))

---

## clu

*Function for extraction of some elements for objects, returned by functions for Generalized blockmodeling*

---

### Description

Function for extraction of clu (partition), all best clus (partitions), IM (image or blockmodel) and err (total error or inconsistency) for objects, returned by functions criticFunC or optRandomParC.
Usage

clu(res, which = 1, ...)  
IM(res, which = 1, drop=TRUE, ...)  
EM(res, which = 1, drop=TRUE, ...)  
err(res, ...)  
partitions(res)

Arguments

res Result of function critFunC or optRandomParC.  
which From which (if there are more than one) "best" solution should the element be extracted. Warning! which greater than the number of "best" partitions produces an error.  
drop If TRUE (default), dimensions that have only one level are dropped (drop function is applied to the final result).  
... Not used.

Value

The desired element.

Author(s)

Aleš Žiberna

References


See Also

critFunC, plot.mat, optRandomParC

Examples

n <- 8 # If larger, the number of partitions increases dramatically,  
# as does if we increase the number of clusters  
net <- matrix(NA, ncol = n, nrow = n)  
clu <- rep(1:2, times = c(3, 5))  
tclu <- table(clu)  
net[clu == 1, clu == 1] <- rnorm(n = tclu[1] * tclu[1], mean = 0, sd = 1)  
net[clu == 1, clu == 2] <- rnorm(n = tclu[1] * tclu[2], mean = 4, sd = 1)  
net[clu == 2, clu == 1] <- rnorm(n = tclu[2] * tclu[1], mean = 0, sd = 1)
```r
net[clu == 2, clu == 2] <- rnorm(n = tclu[2] * tclu[2], mean = 0, sd = 1)

# We select a random partition and then optimize it
all.par <- nkpartitions(n = n, k = length(tclu))
# Forming the partitions
all.par <- lapply(apply(all.par, 1, list), function(x) x[[1]])
# to make a list out of the matrix
res <- optParC(M = net,
               clu = all.par[[sample(1:length(all.par), size = 1)]],
               approaches = "hom", homFun = "ss", blocks = "com")
plot(res) # Hopefully we get the original partition
clu(res) # Hopefully we get the original partition
err(res) # Error
IM(res) # Image matrix/array.
EM(res) # Error matrix/array.
```

---

**critFunC**

*Functions for Generalized blockmodeling for valued networks*

**Description**

Functions for implementation of Generalized blockmodeling for valued networks where the values of the ties are assumed to be measured on at least interval scale. critFunC calculates the criterion function, based on the network, partition and blockmodel/equivalence. optParC optimizes a partition based on the criterion function based on a local search algorithm.

**Usage**

```r
critFunC(M, clu, approaches, blocks, isTwoMode = NULL, isSym = NULL, diag = 1, IM = NULL, EM = NULL, Earr = NULL, justChange = FALSE, rowCluChange = c(0, 0), colCluChange = c(0, 0), sameIM = FALSE, regFun = "max", homFun = "ss", usePreSpecM = NULL, preSpecM = NULL, save.initial.param = TRUE, relWeights = 1, posWeights = 1, blockTypeWeights = 1, combWeights = NULL, returnEnv = FALSE)
```

```r
optParC(M, clu, approaches, blocks, nMode = NULL, isSym = NULL, diag = 1, useMulti = FALSE, maxPar = 50, IM = NULL, EM = NULL, Earr = NULL, justChange = TRUE, minUnitsRowCluster = 1, minUnitsColCluster = 1, maxUnitsRowCluster = 9999, maxUnitsColCluster = 9999, relWeights = 1, posWeights = 1, blockTypeWeights = 1, combWeights = NULL, exchangeClusters = "all", save.initial.param = TRUE)
```
Arguments

M
A matrix representing the (usually valued) network. For multi-relational networks, this should be an array with the third dimension representing the relation. The network can have one or more modes (different kinds of units with no ties among themselves). If the network is not two-mode, the matrix must be square.

clu
A partition. Each unique value represents one cluster. If the network is one-mode, than this should be a vector, else a list of vectors, one for each mode. Similarly, if units are comprised of several sets, clu should be the list containing one vector for each set.

approaches
One of the approaches (for each relation in multi-relational networks in a vector) described in Žiberna (2007). Possible values are:
"bin" - binary blockmodeling,
"val" - valued blockmodeling,
"hom" - homogeneity blockmodeling,
"ss" - sum of squares homogeneity blockmodeling, and
"ad" - absolute deviations homogeneity blockmodeling.

The last two options are "shorthand" for specifying approaches="hom" and homFun to either "ss" or "ad".

blocks
A vector, a list of vectors or an array with names of allowed block types.

Only listing of allowed block types (blockmodel is not pre-specified).
A vector with names of allowed blocktypes. For multi-relational networks, it can be a list of such vectors. For approaches = "bin" or approaches = "val", at least two should be selected. Possible values are:
"nul" - null or empty block
"com" - complete block
"rdo", "cdo" - row and column-dominant blocks (binary and valued approach only)
"reg" - (f-)regular block
"rre", "cre" - row and column-(f-)regular blocks
"rfn", "cfn" - row and column-dominant blocks (binary, valued only)
"den" - density block (binary approach only)
"avg" - average block (valued approach only)
"dnc" - do not care block - the error is always zero
The ordering is important, since if several block types have identical error, the first on the list is selected.

A pre-specified blockmodel.
An array with dimensions four dimensions (see example below). The third and the fourth represent the clusters (for rows and columns). The first is as long as the maximum number of allows block types for a given block. If some block has less possible block types, the empty slots should have values NA. The second dimension is the number of relations (1 for single-relational networks). The values in the array should be the ones from above. The array can have only three dimensions in case of one-relational networks or if the same pre-specified block-model is assumed for all relations. Further, it can have only two dimensions, if
in addition only one block type is allowed per block.

isTwoMode  1 for one-mode networks and 2 for two-mode networks. The default value is set to NULL.

isSym    Specifying if the matrix (for each relation) is symmetric.

diag    Should the special status of diagonal be acknowledged. The default value is set to 1.

IM    The obtained image for objects. For debugging purposes only.

EM    Block errors by blocks. For debugging purposes only.

Earr    The array of errors for all allowed block types by next dimensions: allowed block types, relations, row clusters and column clusters. The dimensions should match the dimensions of the block argument if specified as an array. For debugging purposes only.

justChange    Value specifying if only the errors for changed clusters should be computed. Used only for debugging purposes by developers.

rowCluChange    An array holding the two row clusters where the change occurred. Used only for debugging purposes by developers.

colCluChange    An array holding the column clusters where the change occurred. Used only for debugging purposes by developers.

sameIM    Should we demand the same blockmodel image for all relations. The default value is set to FALSE.

regFun    Function f used in row-f-regular, column-f-regular, and f-regular blocks. Not used in binary approach. For multi-relational networks, it can be a vector of such character strings. The default value is set to "max".

homFun    In case of homogeneity blockmodeling two variability criteria can be used: "ss" - sum of squares (set by default) and "ad" - absolute deviations.

usePreSpecM    Specifying weather a pre-specified value should be used when computing inconsistency.

preSpecM    Sufficient value for individual cells for valued approach. Can be a number or a character string giving the name of a function. Set to "max" for implicit approach. For multi-relational networks, it can be a vector of such values. In case of binary blockmodeling this argument is a threshold used for binerizing the network. Therefore all values with values lower than preSpecM are recoded into 0s, all other into 1s. For multi-relational networks, it can be a vector of such values. In case of pre-specified blockmodeling, it can have the same dimensions as blocks.

save.initial.param    Should the initial parameters (approaches, ...) be saved. The default value is TRUE.

relWeights    Weights for all type of relations in a blockmodel. The default value is set to 1.

posWeights    Weights for positions in the blockmodel (the dimensions must be the same as the error matrix (rows, columns)). For now this is a matrix (two-dimensional) even for multi-relational networks.
**blockTypeWeights**
Weights for each type of block used, if they are to be different across block types (see `blocks` above). It must be supplied in form of a named vector, where the names are one or all allowed block types from `blocks`. If only some block types are specified, the other have a default weight of 1. The default value is set to 1.

**combWeights**
Weights for all type of block used. The default value is set to NULL. The dimension must be the same as `blocks`, if `blocks` would be specified in array format (which is usual in pre-specified case).

**returnEnv**
Should the function also return the environment after its completion.

**useMulti**
Which version of local search should be used. The default value is set to `FALSE`. If `FALSE`, first possible all moves in random order and then all possible exchanges in random order are tried. When a move with lower value of criterion function is found, the algorithm moves to this new partition. If `TRUE` the version of local search where all possible moves and exchanges are tried first and then the one with the lowest error is selected and used. In this case, several optimal partitions are found. `maxPar` best partitions are returned.

**maxPar**
The number of partitions with optimal criterion function to be returned. Only used if `useMulti` is `TRUE`.

**nMode**
Number of nodes. If `NULL`, then determined from `clu`.

**minUnitsRowCluster**
Minimum number of units in row cluster.

**minUnitsColCluster**
Minimum number of units in col cluster.

**maxUnitsRowCluster**
Maximum number of units in row cluster.

**maxUnitsColCluster**
Maximum number of units in col cluster.

**exchangeClusters**
A matrix of dimensions "number of clusters" x "number of clusters" indicating to which clusters can units from a specific cluster be moved. Useful for multi-level blockmodeling or in some other cases where some units cannot mix.

---

**Value**

`critFunC` returns a list containing:

**M**
The matrix of the network analyzed.

**err**
The error or inconsistency empirical network with the ideal network for a given blockmodel (model, approach,...) and partition.

**clu**
The analyzed partition.

**EM**
Block errors by blocks.

**IM**
The obtained image for objects.

**BM**
Block means by block - only for Homogeneity blockmodeling.
**critFunC**

**Err**  The array of errors for all allowed block types by next dimensions: allowed block types, relations, row clusters and column clusters. The dimensions should match the dimensions of the block argument if specified as an array.

optParC returns a list containing:

- **M**  The matrix of the network analyzed.
- **err**  The error or inconsistency empirical network with the ideal network for a given blockmodel (model, approach,...) and partition.
- **clu**  The analyzed partition.
- **EM**  Block errors by blocks.
- **IM**  The obtained image for objects.
- **BM**  Block means by block - only for Homogeneity blockmodeling.
- **Err**  The array of errors for all allowed block types by next dimensions: allowed block types, relations, row clusters and column clusters. The dimensions should match the dimensions of the block argument if specified as an array.
- **useMulti**  The value of the input parameter `useMulti`.
- **bestRowParMatrix**  (If `useMulti = TRUE`) Matrix, where there are different solutions for columns, where rows represent units.
- **sameErr**  The number of partitions with the minimum value of the criterion function.

**Author(s)**

Aleš Žiberna

**References**


**See Also**

- `optRandomParC`, `IM`, `clu`, `err`, `plot.crit.fun`
Examples

# Generating a simple network corresponding to the simple Sum of squares
# Structural equivalence with blockmodel:
# nul com
# nul nul
n <- 20
net <- matrix(NA, ncol = n, nrow = n)
clu <- rep(1:2, times = c(5, 15))
tclu <- table(clu)
net[clu == 1, clu == 1] <- rnorm(n = tclu[1] * tclu[1], mean = 0, sd = 1)
net[clu == 1, clu == 2] <- rnorm(n = tclu[1] * tclu[2], mean = 4, sd = 1)
net[clu == 2, clu == 1] <- rnorm(n = tclu[2] * tclu[1], mean = 0, sd = 1)
net[clu == 2, clu == 2] <- rnorm(n = tclu[2] * tclu[2], mean = 0, sd = 1)

# Computation of criterion function with the correct partition
res <- critFunC(M = net, clu = clu, approaches = "hom", homFun = "ss", blocks = "com")
res$err  # The error is relatively small
res$BM  # The block means are around 0 or 4
plot(res)

# Computation of criterion function with the correct partition and correct pre-specified blockmodel
# Prespecified blockmodel used
# nul com
# nul nul
B <- array(NA, dim = c(1, 1, 2, 2))
B[1, 1, , ] <- "nul"
B[1, 1, 1, 2] <- "com"
res <- critFunC(M = net, clu = clu, approaches = "hom", homFun = "ss", blocks = B)
res$err  # The error is relatively small
res$IM
plot(res)

# Computation of criterion function with the correct partition and pre-specified blockmodel with some alternatives
# Prespecified blockmodel used
# nul nul|com
# nul nul
B <- array(NA, dim = c(2, 2, 2))
B[1, , ] <- "nul"
B[2, 1, 2] <- "com"
res <- critFunC(M = net, clu = clu, approaches = "hom", homFun = "ss", blocks = B)
res$err  # The error is relatively small
res$IM
plot(res)

# Computation of criterion function with random partition
clu.rnd <- sample(1:2, size = n, replace = TRUE)
res.rnd <- critFunC(M = net, clu = clu.rnd, approaches = "hom", homFun = "ss", blocks = "com")
res.rnd$err  # The error is larger
res.rnd$BM  # Random block means
plot(res.rnd)

# Adapt network for Valued blockmodeling with the same model
net[net > 4] <- 4
net[net < 0] <- 0

# Computation of criterion function with the correct partition
res <- critFunC(M = net, clu = clu, approaches = "val",
blocks = c("nul", "com"), preSpecM = 4)
res$err # The error is relatively small
res$IM
# The image corresponds to the one used for generation of
# The network
plot(res)

# Computation of criterion function with random partition
res.rnd <- critFunC(M = net, clu = clu.rnd, approaches = "val",
blocks = c("nul", "com"), preSpecM = 4)
res.rnd$err # The error is larger
res.rnd$IM # All blocks are probably nul
plot(res.rnd)

---

**find.m**

*Computing the threshold*

### Description

The functions compute the maximum value of $m/cut$ where a certain block is still classified as `alt.blocks` and not "null". The difference between `find.m` and `find.m2` is that `find.m` uses an optimization approach and is faster and more precise than `find.m2`. However, `find.m` only supports regular ("reg") and complete ("com") as `alt.blocks`, while `find.m2` supports all block types. Also, `find.m` does not always work, especially if `cormet` is not "none".

### Usage

```r
find.m(M, clu, alt.blocks = "reg", diag = !is.list(clu),
cormet = "none", half = TRUE, FUN = "max")
find.m2(M, clu, alt.blocks = "reg", neval = 100, half = TRUE,
ms = NULL, ...)
find.cut(M, clu, alt.blocks = "reg", cuts = "all", ...)
```

### Arguments

- **M**
  
  A matrix representing the (usually valued) network. For now, only one-relational networks are supported. The network can have one or more modes (different kinds of units with no ties among themselves. If the network is not two-mode, the matrix must be square.

- **clu**
  
  A partition. Each unique value represents one cluster. If the network is one-mode, then this should be a vector, else a list of vectors, one for each mode.
alt.blocks Only one of allowed blocktypes, as alternative to the null block:
  "com" - complete block
  "rdo", "cdo" - row and column-dominant blocks (binary, valued, and implicit approach only)
  "reg" - (f-)regular block
  "rre", "cre" - row and column-(f-)regular blocks
  "rfn", "cfn" - row and column-dominant blocks (binary, valued, and implicit approach only)
  "den" - density block (binary approach only)
  "avg" - average block (valued approach only)

diag (default = TRUE) Should the special status of diagonal be acknowledged.
cormet Which method should be used to correct for different maximum error contributions
  "none" - no correction
  "censor" - censor values larger than M
  "correct" - so that the maximum possible error contribution of the cell is the same regardless of a condition (either that something must be 0 or at least M).

FUN (default = "max") Function f used in row-f-regular, column-f-regular, and f-regular blocks.
cuts The cuts, which should be evaluated. If cuts="all" (default), all unique values are evaluated.
neval A number of different m values to be evaluated.
half Should the returned value of m be one half of the value where the inconsistencies are the same.
ms The values of m where the function should be evaluated.
... Other parameters to crit.fun.

Value
A matrix of maximal m/cut values.

Author(s)
Aleš Žiberna

References
formatA

Description

Formats a vector or matrix of numbers so that all have equal length (digits). This is especially suitable for printing tables.

Usage

formatA(x, digits = 2, FUN = round, ...)

Arguments

x  
A numerical vector or matrix.

digits  
The number of desired digits.

FUN  
Function used for "shortening" the numbers.

...  
Additional arguments to format.

Value

A character vector or matrix.

Author(s)

Aleš Žiberna

See Also

find.m, find.m2, find.cut

Examples

A <- matrix(c(1, 1.02002, 0.2, 10.3), ncol = 2)
formatA(A)
fun.by.blocks  

Computation of function values by blocks

Description

Computes a value of a function over blocks of a matrix, defined by a partition.

Usage

fun.by.blocks(x, ...)

## Default S3 method:
fun.by.blocks(x = M, M = x, clu, 
ignore.diag = "default", sortNames = TRUE, 
FUN = "mean", ...)

## S3 method for class 'opt.more.par'
fun.by.blocks(x, which = 1, ...)

Arguments

x  An object of suitable class or a matrix representing the (usually valued) network. For now, only one-relational networks are supported. The network can have one or more modes (different kinds of units with no ties among themselves. If the network is not two-mode, the matrix must be square.

M  A matrix representing the (usually valued) network. For now, only one-relational networks are supported. The network can have one or more modes (different kinds of units with no ties among themselves. If the network is not two-mode, the matrix must be square.

clu  A partition. Each unique value represents one cluster. If the network is one-mode, then this should be a vector, else a list of vectors, one for each mode.

ignore.diag  Should the diagonal be ignored.

sortNames  Should the rows and columns of the matrix be sorted based on their names.

FUN  The function to be computed over the blocks.

which  Which (if several) of the "best" solutions should be used.

...  Further arguments to fun.by.blocks.default.

Value

A numerical matrix of FUN values by blocks, induced by a partition clu.

Author(s)

Aleš Žiberna
genMatrixMult

References


See Also

optRandomParC, optParC

Examples

n <- 8 # If larger, the number of partitions increases dramatically, # as does if we increase the number of clusters net <- matrix(NA, ncol = n, nrow = n) clu <- rep(1:2, times = c(3, 5)) tclu <- table(clu) net[clu == 1, clu == 1] <- rnorm(n = tclu[1] * tclu[1], mean = 0, sd = 1) net[clu == 1, clu == 2] <- rnorm(n = tclu[1] * tclu[2], mean = 4, sd = 1) net[clu == 2, clu == 1] <- rnorm(n = tclu[2] * tclu[1], mean = 0, sd = 1) net[clu == 2, clu == 2] <- rnorm(n = tclu[2] * tclu[2], mean = 0, sd = 1)

# Optimizing 10 random partitions with optRandomParC res <- optRandomParC(M = net, k = 2, rep = 10, approaches = "hom", homFun = "ss", blocks = "com") plot(res) # Hopefully we get the original partition fun.by.blocks(res) # Computing mean by blocks, ignoring the diagonal (default)

---

genMatrixMult  

*Generalized matrix multiplication*

**Description**

Computes a generalized matrix multiplication, where sum and product functions (element-wise and summary functions) can be replaced by arbitrary functions.

**Usage**

genMatrixMult(A, B, FUNelement = "*", FUNsummary = sum)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>The first matrix.</td>
</tr>
<tr>
<td>B</td>
<td>The second matrix.</td>
</tr>
<tr>
<td>FUNelement</td>
<td>Element-wise operator.</td>
</tr>
<tr>
<td>FUNsummary</td>
<td>Summary function.</td>
</tr>
</tbody>
</table>
The function for generating random partitions

The function generates random partitions. The function is meant to be called by the function `optRandomParC`.

Usage

```r
genRandomPar(k, n, seed = NULL, mingr = 1, maxgr = Inf,
             addParam = list(genPajekPar = TRUE, probGenMech = NULL))
```

Arguments

- **k**: Number of clusters (by modes).
- **n**: Number of units (by modes).
- **seed**: Seed for generating random numbers (partitions).
- **mingr**: Minimal allowed group size.
- **maxgr**: Maximal allowed group size.
This has to be a list with the following parameters (any or all can be missing, then the default values (see usage) are used):

- `genPajekPar` - Should the partitions be generated as in Pajek (Batagelj & Mrvar, 2006). If FALSE, all partitions are selected completely at random while making sure that the partitions have the required number of clusters.

- `probGenMech` - Here the probabilities for 4 different generating mechanisms can be specified. If this is not specified, the value is set to \( \{1/3, 1/3, 1/3, 0\} \) if `genPajekPar` is TRUE and to \( \{0, 0, 0, 1\} \) if `genPajekPar` is FALSE. The first 3 mechanisms are the same as implemented in Pajek (the second one has almost all units in only one cluster) and the fourth is completely random (from uniform distribution).

Value

A random partition in the format required by `optRandomParC`. If a network has several modes, then a list of partitions, one for each mode.

Author(s)

Aleš Žiberna

References


Description

The function calls function `gplot` from the library `sna` with different defaults. Use `fun` for plotting image graphs.

Usage

gplot1(M, diag = TRUE,
     displaylabels = TRUE, boxed.labels = FALSE,
     loop.cex = 4, edge.lwd = 1, edge.col = "default",
     rel.thresh = 0.05, ...)

gplot2(M, uselen = TRUE, usecurve = TRUE,
     edge.len = 0.001, diag = TRUE,
     displaylabels = TRUE, boxed.labels = FALSE,
     loop.cex = 4, arrowhead.cex = 2.5,
     edge.lwd = 1, edge.col = "default", rel.thresh = 0.05, ...)
Arguments

M
A matrix (array) of a graph or set thereof. This data may be valued.

diag
Boolean indicating whether or not the diagonal should be treated as valid data. Set this TRUE if and only if the data can contain loops. diag is FALSE by default.

rel.thresh
Real number indicating the lower relative (compared to the highest value) threshold for tie values. Only ties of value thresh are displayed. By default, thresh = 0.

displaylabels
Boolean; should vertex labels be displayed.

boxed.labels
Boolean; place vertex labels within boxes.

arrowhead.cex
An expansion factor for edge arrowheads.

loop.cex
Expansion factor for loops; may be given as a vector, if loops are to be of different sizes.

displaylabels
Color for edges; may be given as a vector or adjacency matrix, if edges are to be of different colors.

data
Line width scale for edges; if set greater than 0, edge widths are scaled by edge.lwd*dat. May be given as a vector or adjacency matrix, if edges are to have different line widths.

if uselen == TRUE, curved edge lengths are scaled by edge.len.

usecurve
Boolean; should we use edge.curve.

... Additional arguments to plot or gplot from package sna:

mode: the vertex placement algorithm; this must correspond to a gplot.layout function from package sna.

Value
Plots a graph.

Author(s)
Aleš Žiberna

See Also
sna:gplot
ircNorm

Function for iterated row and column normalization of valued matrices

Description

The aim is to obtain a matrix with row and column sums equal to 1. This is achieved by iterating row and column normalization. This is usually not possible if any row or column has only 1 non-zero cell.

Usage

ircNorm(M, eps = 10^-12, maxiter = 1000)

Arguments

M A non-negative valued matrix to be normalized.
eps The maximum allows squared deviation of a row or column’s maximum from 1 (if not exactly 0). Also, if the all deviations in two consecutive iterations are smaller, the process is terminated.
maxiter Maximum number of iterations. If reached, the process is terminated and the current solution returned.

Value

Normalized matrix.

Author(s)

Aleš Žiberna

Examples

A <- matrix(runif(100), ncol = 10)
A # A non-normalized matrix with different row and column sums.
apply(A, 1, sum)
apply(A, 2, sum)
A.norm <- ircNorm(A)
A.norm # Normalized matrix with all row and column sums approximately 1.
apply(A.norm, 1, sum)
apply(A.norm, 2, sum)
Description

The function `nkpartitions` lists all possible partitions of n objects in to k clusters. The function `nkpar` only gives the number of such partitions.

Usage

```
nkpartitions(n, k, exact = TRUE, print = FALSE)
nkpar(n, k)
```

Arguments

- `n`: Number of units/objects.
- `k`: Number of clusters/groups.
- `exact`: Search for partitions with exactly k or at most k clusters.
- `print`: Print results as they are found.

Value

The matrix or number of possible partitions.

Author(s)

Chris Andrews

Examples

```
n <- 8 # If larger, the number of partitions increases dramatically,
# as does if we increase the number of clusters
net <- matrix(NA, ncol = n, nrow = n)
clu <- rep(1:2, times = c(3, 5))
tclu <- table(clu)
et[clu == 1, clu == 1] <- rnorm(n = tclu[1] * tclu[1], mean = 0, sd = 1)
et[clu == 1, clu == 2] <- rnorm(n = tclu[1] * tclu[2], mean = 4, sd = 1)
et[clu == 2, clu == 1] <- rnorm(n = tclu[2] * tclu[1], mean = 0, sd = 1)
et[clu == 2, clu == 2] <- rnorm(n = tclu[2] * tclu[2], mean = 0, sd = 1)

# Computation of criterion function with the correct partition
nkpar(n = n, k = length(tclu)) # Computing the number of partitions
all.par <- nkpartitions(n = n, k = length(tclu)) # Forming the partitions
all.par <- lapply(apply(all.par, 1, list), function(x) x[[1]])
# to make a list out of the matrix
res <- critFunC(M = net, clu = clu, approaches = "val",
    blocks = c("nul", "com"), preSpecM = 4)
plot(res) # We get the original partition
```
notesBorrowing

The notes borrowing network between social-informatics students

Description
The data come from a survey conducted in May 1993 on 13 social-informatics students (Hlebec, 1996). The network was constructed from answers to the question, “How often did you borrow notes from this person?” for each of the fellow students. The respondents indicated the frequency of borrowing by choosing (on a computer) a line of length 1–20, where 1 meant no borrowing. 1 was deducted from all answers, so that 0 now means no borrowing. The data was first used for blockmodeling in Žiberna (2007).

Usage
data("notesBorrowing")

Format
The data set is a valued matrix with 13 rows and columns.

References

Examples
data(notesBorrowing)

# Plot the network.
# (The function plotMat is from blockmodeling package.)
# plotMat(nyt)

optRandomParC

Optimizing a set of partitions based on the value of a criterion function

Description
The function optimizes a set of partitions based on the value of a criterion function (see critFunC for details on the criterion function) for a given network and blockmodel for Generalized block-modeling (Žiberna, 2007) based on other parameters (see below). The optimization is done through local optimization, where the neighborhood of a partition includes all partitions that can be obtained by moving one unit from one cluster to another or by exchanging two units (from different clusters). A list of partitions can or the number of clusters and a number of partitions to generate can be specified (optParC).
optRandomParC(M, k, approaches, blocks, rep, save.initial.param = TRUE, save.initial.param.opt = FALSE, deleteMs = TRUE, max.iden = 10, switch.names = NULL, return.all = FALSE, return.err = TRUE, seed = NULL, RandomSeed = NULL, parGenFun = genRandomPar, mingr = NULL, maxgr = NULL, addParam = list(genPajekPar = TRUE, probGenMech = NULL), maxTriesToFindNewPar = rep * 10, skip.par = NULL, useOptParMultiC = FALSE, useMulti = useOptParMultiC, printRep = ifelse(rep <= 10, 1, round(rep/10)), n = NULL, nCores = 1, ...)

Arguments

M A matrix representing the (usually valued) network. For multi-relational networks, this should be an array with the third dimension representing the relation. The network can have one or more modes (different kinds of units with no ties among themselves). If the network is not two-mode, the matrix must be square.

k The number of clusters used in the generation of partitions.

approaches One of the approaches (for each relation in multi-relational networks in a vector) described in Žiberna (2007). Possible values are: "bin" - binary blockmodeling, "val" - valued blockmodeling, "hom" - homogeneity blockmodeling, "ss" - sum of squares homogeneity blockmodeling, and "ad" - absolute deviations homogeneity blockmodeling.

blocks A vector, a list of vectors or an array with names of allowed block types.

The last two options are "shorthand" for specifying approaches="hom" and homFun to either "ss" or "ad".

A matrix representing the (usually valued) network. For multi-relational networks, this should be an array with the third dimension representing the relation. The network can have one or more modes (different kinds of units with no ties among themselves). If the network is not two-mode, the matrix must be square.

The number of clusters used in the generation of partitions.

A vector, a list of vectors or an array with names of allowed block types.

Only listing of allowed block types (blockmodel is not pre-specified).

A vector with names of allowed blocktypes. For multi-relational networks, it can be a list of such vectors. For approaches = "bin" or approaches = "val", at least two should be selected. Possible values are: "nul" - null or empty block "com" - complete block "rdo", "cdo" - row and column-dominant blocks (binary and valued approach only) "reg" - (f-)regular block "rre", "cre" - row and column-(f-)regular blocks "rfn", "cfn" - row and column-dominant blocks (binary, valued only) "den" - density block (binary approach only) "avg" - average block (valued approach only)
"dnc" - do not care block - the error is always zero
The ordering is important, since if several block types have identical error, the first on the list is selected.

A pre-specified blockmodel.
An array with dimensions four dimensions (see example below). The third and the fourth represent the clusters (for rows and columns). The first is as long as the maximum number of allows block types for a given block. If some block has less possible block types, the empty slots should have values NA. The second dimension is the number of relations (1 for single-relational networks). The values in the array should be the ones from above. The array can have only three dimensions in case of one-relational networks or if the same pre-specified block-model is assumed for all relations. Further, it can have only two dimensions, if in addition only one block type is allowed per block.

rep
The number of repetitions/different starting partitions to check.

save.initial.param
Should the initial parameters (approaches, ...) be saved. The default value is TRUE.

save.initial.param.opt
Should the initial parameters (approaches, ...) of using optParC be saved. The default value is FALSE.

deleteMs
Delete networks/matrices from the results of to save space.

max.iden
Maximum number of results that should be saved (in case there are more than max.iden results with minimal error, only the first max.iden will be saved).

switch.names
Should partitions that only differ in group names be considered equal.

return.all
If FALSE, solution for only the best (one or more) partition/s is/are returned.

return.err
Should the error for each optimized partition be returned.

seed
Optional. The seed for random generation of partitions.

RandomSeed
Optional. Integer vector, containing the random number generator. It is only looked for in the user's workspace.

parGenFun
The function (object) that will generate random partitions. The default function is genRandomPar. The function has to accept the following parameters: k (number of partitions by modes, n (number of units by modes), seed (seed value for random generation of partition), addParam (a list of additional parameters).

mingr
Minimal allowed group size.

maxgr
Maximal allowed group size.

addParam
A list of additional parameters for function specified above. In the usage section they are specified for the default function genRandomPar.

genPajekPar
Should the partitions be generated as in Pajek.

probGenMech
Should the probabilities for different mechanisms for specifying the partitions be set. If probGenMech is not set, it is determined based on the parameter genPajekPar.

maxTriesToFindNewPar
The maximum number of partition try when trying to find a new partition to optimize that was not yet checked before - the default value is rep * 1000.
skip.par  The partitions that are not allowed or were already checked and should therefore be skipped.

useOptParMultiC  For backward compatibility. May be removed soon. See next argument.

useMulti  Which version of local search should be used. Default is currently FALSE. If FALSE, first possible all moves in random order and then all possible exchanges in random order are tried. When a move with lower value of criterion function is found, the algorithm moves to this new partition. If TRUE the version of local search where all possible moves and exchanges are tried first and then the one with the lowest error is selected and used. In this case, several optimal partitions are found. maxPar best partitions are returned.

printRep  Should some information about each optimization be printed.

n  The number of units by "modes". It is used only for generating random partitions. It has to be set only if there are more than two modes or if there are two modes, but the matrix representing the network is one mode (both modes are in rows and columns).

nCores  Number of cores to be used. Value 0 means all available cores. It can also be a cluster object.

...  Arguments passed to other functions, see critFunC.

Value

M  The matrix of the network analyzed.

res  If return.all = TRUE - A list of results the same as best - one best for each partition optimized.

best  A list of results from crit.fun.tmp with the same elements as the result of crit.fun, only without M.

err  If return.err = TRUE - The vector of errors or inconsistencies of the empirical network with the ideal network for a given blockmodel (model,approach,...) and partitions.

nIter  The vector of the number of iterations used - one value for each starting partition that was optimized. It can show that maxIter is too low if a lot of these values have the value of maxIter.

checked.par  If selected - A list of checked partitions. If merge.save.skip.par is TRUE, this list also includes the partitions in skip.par.

call  The call used to call the function.

initial.param  If selected - The initial parameters are used.

Warning

It should be noted that the time complexity of package blockmodeling is increasing with the number of units and the number of clusters (due to its algorithm). Therefore the analysis of network with more than 100 units can take a lot of time (from a few hours to a few days).
Author(s)
Aleš Žiberna

References

See Also
critFunC

Examples
n <- 8 # If larger, the number of partitions increases dramatically
# as does if we increase the number of clusters
net <- matrix(NA, ncol = n, nrow = n)
clu <- rep(1:2, times = c(3, 5))
tclu <- table(clu)
net[clu == 1, clu == 1] <- rnorm(n = tclu[1] * tclu[1], mean = 0, sd = 1)
net[clu == 1, clu == 2] <- rnorm(n = tclu[1] * tclu[2], mean = 4, sd = 1)
net[clu == 2, clu == 1] <- rnorm(n = tclu[2] * tclu[1], mean = 0, sd = 1)
net[clu == 2, clu == 2] <- rnorm(n = tclu[2] * tclu[2], mean = 0, sd = 1)

# We select a random partition and then optimize it
all.par <- nkpartitions(n = n, k = length(tclu))
# Forming the partitions
all.par <- lapply(apply(all.par, 1, list), function(x)x[[1]])

# Optimizing one partition
res <- optParC(M = net,
clu = all.par[[sample(1:length(all.par), size = 1)]],
approaches = "hom", homFun = "ss", blocks = "com")
plot(res) # Hopefully we get the original partition

# Optimizing 10 random chosen partitions with optRandomParC
res <- optRandomParC(M = net, k = 2, rep = 10,
approaches = "hom", homFun = "ss", blocks = "com")
plot(res) # Hopefully we get the original partition
Functions for loading and writing Pajek files

Description

Functions for reading/loading and writing Pajek files:
- loadnetwork - Loads a Pajek ".net" filename as a matrix. For now, only simple one and two-mode networks are supported (eg. only single relations, no time information).
- loadnetwork2 - The same as above, but adapted to be called within loadpajek.
- loadnetwork3 - Another version for reading networks.
- loadnetwork4 - Another version for reading networks.
- savenetwork - Saves a matrix into a Pajek ".net" filename.
- loadmatrix - Loads a Pajek ".mat" filename as a matrix.
- savematrix - Saves a matrix into a Pajek ".mat" filename.
- loadvector - Loads a Pajek ".clu" filename as a vector.
- loadvector2 - The same as above, but adapted to be called within loadpajek - as a consequence not suited for reading clusters.
- savevector - Saves a vector into a Pajek ".clu" filename.
- loadpajek - Loads a Pajek project file name (".paj") as a list with the following components: Networks, Partitions, Vectors and Clusters. Clusters and hierarchies are dismissed.

Usage

loadnetwork(filename, useSparseMatrix = NULL, minN = 50)
loadnetwork2(filename, useSparseMatrix = NULL, minN = 50, safe = TRUE, closeFile = TRUE)
loadnetwork3(filename, useSparseMatrix = NULL, minN = 50)
loadnetwork4(filename, useSparseMatrix = NULL, minN = 50, fill = FALSE)
savenetwork(n, filename, twomode = "default", symmetric = NULL)
loadmatrix(filename)
savematrix(n, filename, twomode = 1)
loadvector(filename)
loadvector2(filename)
savevector(v, filename)
loadpajek(filename)

Arguments

filename
The name of the filename to be loaded or saved to or an open file object.
useSparseMatrix
Should a sparse matrix be use instead of the ordinary one? Sparse matrices can only be used if package Matrix is installed. The default NULL uses sparse matrices for networks with more that minN vertices.
**plot.mat**

- **minN**: The minimal number of units in the network to use sparse matrices.
- **n**: A matrix representing the network.
- **twomode**: 1 for one-mode networks and 2 for two-mode networks. Default sets the argument to 1 for square matrices and to 2 for others.
- **symetric**: If TRUE, only the lower part of the matrix is used and the values are interpreted as "Edges", not "Arcs".
- **v**: A vector.
- **fill**: If TRUE, then in case the rows have unequal length, blank fields are added.
- **safe**: If FALSE error will occur if not all vertices have labels. If TRUE reading works faster.
- **closeFile**: Should the connection be closed at the end. Should be always TRUE if function is used directly.

**Value**

NULL, a matrix or a vector (see Description).

**Author(s)**

Vladimir Batagelj & Andrej Mrvar (most functions), Aleš Žiberna (loadnetwork, loadpajek and modification of others)

**References**


**See Also**

plot.mat, critFunC, optRandomParC

---

**Description**

The main function plot.mat or plotMat plots a (optionally partitioned) matrix. If the matrix is partitioned, the rows and columns of the matrix are rearranged according to the partitions. Other functions are only wrappers for plot.mat or plotMat for convenience when plotting the results of the corresponding functions. The plotMatNm plots two matrices based on M, normalized by rows and columns, next to each other. The plot.array or plotArray plots an array. plot.mat.nm has been replaced by plotMatNm.
Usage

plotMat(x = M, M = x, clu = NULL,
        ylab = "", xlab = "", main =
        NULL, print.val = !length(table(M)) <= 2, print.0 =
        FALSE, plot.legend = !print.val && !length(table(M))
        <= 2, print.legend.val = "out", print.digits.legend =
        2, print.digits.cells = 2, print.cells.mf = NULL,
        outer.title = FALSE, title.line = ifelse(outer.title,
        -1.5, 7), mar = c(0.5, 7, 8.5, 0) + 0.1, cex.val =
        "default", val.y.coor.cor = 0, val.x.coor.cor = 0,
        cex.legend = 1, legend.title = "Legend", cex.axes =
        "default", print.axes.val = NULL, print.x.axis.val =
        !is.null(colnames(M)), print.y.axis.val =
        !is.null(rownames(M)), x.axis.val.pos = 1.01,
        y.axis.val.pos = -0.01, cex.main = par()$cex.main,
        cex.lab = par()$cex.lab, yaxis.line = -1.5, xaxis.line
        = -1, legend.left = 0.4, legend.up = 0.03, legend.size
        = 1/min(dim(M)), legend.text.hor.pos = 0.5,
        par.line.width = 3, par.line.col = "blue", IM.dens =
        NULL, print.legend = NULL, print.digits.legend =
        2, print.digits.cells = 2, print.cells.mf = NULL,
        outer.title = FALSE, title.line = ifelse(outer.title,
        -1.5, 7), mar = c(0.5, 7, 8.5, 0) + 0.1, cex.val =
        "default", val.y.coor.cor = 0, val.x.coor.cor = 0,
        cex.legend = 1, legend.title = "Legend", cex.axes =
        "default", print.axes.val = NULL, print.x.axis.val =
        !is.null(colnames(M)), print.y.axis.val =
        !is.null(rownames(M)), x.axis.val.pos = 1.01,
plotMatNm(x = M, M = x, ..., main.title = NULL,
    title.row = "Row normalized",
    title.col = "Column normalized",
    main.title.line = -2, par.set = list(mfrow = c(1, 2))
)

plotArray(x = M, M = x, IM = NULL, ..., main.title = NULL, main.title.line
    = -2, mfrow = NULL)

## S3 method for class 'mat'
plot(x = M, M = x, clu = NULL,
    ylab = "", xlab = "", main = NULL, print.val = is.null(table(M)) <= 2, print.0 = FALSE,
    plot.legend = !length(table(M)) <= 2, print.digits.legend = 2, print.digits.cells = 2,
    outer.title = FALSE, title.line = ifelse(outer.title,
        -1.5, 7), mar = c(0.5, 7, 8.5, 0) + 0.1, cex.val = "default",
    val.y.coor.cor = 0, val.x.coor.cor = 0,
    cex.legend = 1, legend.title = "Legend", cex.axes = "default",
    print.axes.val = NULL, print.x.axis.val = !is.null(colnames(M)), print.y.axis.val =
        !is.null(rownames(M)), x.axis.val.pos = 1.01,
    y.axis.val.pos = -0.01, cex.main = par()$cex.main,
    cex.lab = par()$cex.main, yaxis.line = -1.5, xaxis.line = -1, legend.left = 0.4, legend.up = 0.03, legend.size
= 1/min(dim(M)), legend.text.hor.pos = 0.5, par.line.width = 3, par.line.col = "blue", IM.dens = NULL, IM = NULL, wnet = NULL, wIM = NULL, use.IM = length(dim(IM)) == length(dim(M)) | !is.null(wIM), dens.leg = c(null = 100, nul = 100), blackdens = 70, plotLines = FALSE, frameMatrix = TRUE, x0ParLine = -0.1, x1ParLine = 1, y0ParLine = 0, y1ParLine = 1.1, colByUnits = NULL, colByRow = NULL, colByCol = NULL, mulCol = 2, joinColOperator = "+", colTies = FALSE, maxValPlot = NULL, printMultipliedMessage = TRUE, replaceNAdiagWith0 = TRUE, colLabels = FALSE, ...

## S3 method for class 'crit.fun'
plot(x, main = NULL, ...)  

## S3 method for class 'array'
plot(x = M, M = x, IM = NULL, ..., main.title = NULL, main.title.line = -2, mfrow = NULL)  

## S3 method for class 'opt.par'
plot(x, main = NULL, which = 1, ...)  

## S3 method for class 'opt.par.mode'
plot(x, main = NULL, which = 1, ...)  

## S3 method for class 'opt.more.par'
plot(x, main = NULL, which = 1, ...)  

## S3 method for class 'opt.more.par.mode'
plot(x, main = NULL, which = 1, ...)  

Arguments

x
A result from a corresponding function or a matrix or similar object representing a network.

M
A matrix or similar object representing a network - either x or M must be supplied - both are here to make the code compatible with generic and with older functions.

clu
A partition. Each unique value represents one cluster. If the network is one-mode, then this should be a vector, else a list of vectors, one for each mode.

ylab
Label for y axis.

xlab
Label for x axis.

main
Main title.

main.title
Main title in plot.array version.

main.title.line
The line in which main title is printed in plot.array version.
mfrow  
Arguments to par - number of row and column plots to be plotted on one figure.

print.val  
Should the values be printed in the matrix.

print.0  
If print.val = TRUE Should the 0s be printed in the matrix.

plot.legend  
Should the legend for shades be plotted.

print.legend.val  
Should the values be printed in the legend.

print.digits.legend  
The number of digits that should appear in the legend.

print.digits.cells  
The number of digits that should appear in the cells (of the matrix and/or legend).

print.cells.mf  
If not NULL, the above argument is ignored, the cell values are printed as the cell are multiplied by this factor and rounded.

outer.title  
Should the title be printed on the 'inner' or 'outer' margin of the plot, default is 'inner' margin.

title.line  
The line (from the top) where the title should be printed. The suitable values depend heavily on the displayed type.

mar  
A numerical vector of the form c(bottom, left, top, right) which gives the lines of margin to be specified on the four sides of the plot. The R default for ordinary plots is c(5, 4, 4, 2) + 0.1, while this function default is c(0.5, 7, 8.5, 0) + 0.1.

cex.val  
The size of the values printed. The default is 10 / 'number of units'.

val.y.coor.cor  
Correction for centering the values in the squares in y direction.

val.x.coor.cor  
Correction for centering the values in the squares in x direction.

cex.legend  
The size of the text in the legend.

legend.title  
The title of the legend.

cex.axes  
The size of the characters in axes. Default makes the cex so small that all categories can be printed.

print.axes.val  
Should the axes values be printed. Default prints each axis if rownames or colnames is not NULL.

print.x.axis.val  
Should the x axis values be printed. Default prints each axis if rownames or colnames is not NULL.

print.y.axis.val  
Should the y axis values be printed. Default prints each axis if rownames or colnames is not NULL.

x.axis.val.pos  
The x coordinate of the y axis values.

y.axis.val.pos  
The y coordinate of the x axis values.

cex.main  
The size of the text in the main title.

cex.lab  
The size of the text in matrix.

yaxis.line  
The position of the y axis (the argument 'line').

xaxis.line  
The position of the x axis (the argument 'line').
legend.left  How much left should the legend be from the matrix.
legend.up    How much up should the legend be from the matrix.
legend.size  Relative legend size.
legend.text.hor.pos  Horizontal position of the legend text (bottom) - 0 = bottom, 0.5 = middle,...
par.line.width The width of the line that separates the partitions.
par.line.col  The color of the line that separates the partitions.
IM.dens      The density of shading lines in each block.
IM           The image (as obtained with critFunC) of the blockmodel. dens.leg is used to translate this image into IM.dens.
wnet     Specifies which matrix (if more) should be plotted - used if M is an array.
wIM       Specifies which IM (if more) should be used for plotting. The default value is set to wnet) - used if IM is an array.
use.IM     Specifies if IM should be used for plotting.
dens.leg   It is used to translate the IM into IM.dens.
blackdens  At which density should the values on dark colors of lines be printed in white.
plotLines  Should the lines in the matrix be printed. The default value is set to FALSE, best set to TRUE for very small networks.
frameMatrix Should the matrix be framed (if plotLines is FALSE). The default value is set to TRUE.
x0ParLine  Coordinates for lines separating clusters.
x1ParLine  Coordinates for lines separating clusters.
y0ParLine  Coordinates for lines separating clusters.
y1ParLine  Coordinates for lines separating clusters.
colByUnits Coloring units. It should be a vector of unit length.
colByRow   Coloring units by rows. It should be a vector of unit length.
colByCol   Coloring units by columns. It should be a vector of unit length.
mulCol     Multiply color when joining with row, column. Only used when when colByUnits is not NULL.
joinColOperator Function to join colByRow and colByCol. The default value is set to "+".
colTies     If TRUE, ties are colored, if FALSE, 0-ties are colored.
maxValPlot  The value to use as a maximum when computing colors (ties with maximal positive value are plotted as black).
printMultipliedMessage Should the message '* all values in cells were multiplied by' be printed on the plot. The default value is set to TRUE.
replaceNAdiagWith0 If replaceNAdiagWith0 = TRUE Should the NA values on the diagonal of a matrix be replaced with 0s.
plot.mat

**title.row**  Title for the row-normalized matrix in nm version

**title.col**  Title for the column-normalized matrix in nm version

**par.set**  A list of possible plotting parameters (to par) to be used in nm version

**which**  Which (if there are more than one) of optimal solutions to plot.

**colLabels**  Should the labels of units be colored. If FALSE, these are not colored, if TRUE, they are colored with colors of clusters as defined by palette. This can be also a vector of colors (or integers) for one-mode networks or a list of two such vectors for two-mode networks.

...  Additional arguments to plot.default for plotMat and also to plotMat for other functions.

**Value**

The functions are used for their side effect - plotting.

**Author(s)**

Aleš Žiberna

**References**


**See Also**

critFunC, optRandomParC

**Examples**

```r
# Generation of the network
n <- 20
net <- matrix(NA, ncol = n, nrow = n)
clu <- rep(1:2, times = c(5, 15))
tclu <- table(clu)
net[clu == 1, clu == 1] <- rnorm(n = tclu[1] * tclu[1], mean = 0, sd = 1)
net[clu == 1, clu == 2] <- rnorm(n = tclu[1] * tclu[2], mean = 4, sd = 1)
net[clu == 2, clu == 1] <- rnorm(n = tclu[2] * tclu[1], mean = 0, sd = 1)
net[clu == 2, clu == 2] <- rnorm(n = tclu[2] * tclu[2], mean = 0, sd = 1)

# Ploting the network
plotMat(M = net, clu = clu, print.digits.cells = 3)
class(net) <- "mat"
plot(net, clu = clu)
# See corresponding functions for examples for other plotting
# functions
# presented, that are essentially only the wrappers for "plot.max"
```
**Description**

Rand Index and Rand Index corrected/adjusted for chance for comparing partitions (Hubert & Arabie, 1985). The names of the clusters do not matter.

**Usage**

```r
rand(tab)
rand2(clu1, clu2)
crand(tab)
crand2(clu1, clu2)
```

**Arguments**

- `clu1, clu2` The two partitions to be compared, given in the form of vectors, where for each unit a cluster membership is given.
- `tab` A contingency table obtained as a table(clu1, clu2).

**Value**

The value of Rand Index (corrected/adjusted for chance).

**Author(s)**

Aleš Žiberna

**References**


---

**Description**

Recodes values in a vector.

**Usage**

```r
recode(x, oldcode = sort(unique(x)), newcode)
```
Arguments

x
oldcode
newcode

A vector.
A vector of old codes.
A vector of new codes.

Value

A recoded vector.

Author(s)

Aleš Žiberna

Examples

x <- rep(1:3, times = 1:3)
newx <- recode(x, oldcode = 1:3, newcode = c("a", "b", "c"))

Description

REGE - Algorithms for computing (dis)similarities in terms of regular equivalence (White & Reitz, 1983):

REGE, REGE.for - Classical REGE or REGGE, as also implemented in Ucinet. Similarities in terms of regular equivalence are computed. The REGE.for is a wrapper for calling the FORTRAN subroutine written by White (1985a), modified to be called by R. The REGE does the same, however it is written in R. The functions with and without ".for" differ only in whether they are implemented in R of FORTRAN. Needless to say, the functions implemented in FORTRAN are much faster.

REGE.ow, REGE.ow.for - The above function, modified so that a best match is searched for each arc separately (and not for both arcs, if they exist, together).

REGE.ownm.for - REGE or REGGE, modified to use row and column normalized matrices instead of the original matrix.

REGE.ownm.for - The above function, modified so that a best match for an outgoing ties is searched on row-normalized network and for incoming ties on column-normalized network.

REGD.for - REGD or REGDI, a dissimilarity version of the classical REGE or REGGE. Dissimilarities in terms of regular equivalence are computed. The REGD.for is a wrapper for calling the FORTRAN subroutine written by White (1985b), modified to be called by R.

REGE.FC - Actually an earlier version of REGE. The difference is in the denominator. See Žiberna (2007) for details.

REGE.FC.ow - The above function, modified so that a best match is searched for each arc separately (and not for both arcs, if they exist, together).

other - still in testing stage.
Usage

REGE(M, E = 1, iter = 3, until.change = TRUE, use.diag = TRUE)
REGE.for(M, iter = 3, E = 1)
REGE.nm.for(M, iter = 3, E = 1)
REGE.ow(M, E = 1, iter = 3, until.change = TRUE, use.diag = TRUE)
REGE.ow.for(M, iter = 3, E = 1)
REGE.ownm.for(M, iter = 3, E = 1)
REGD.for(M, iter = 3, E = 0)
REGD.ow.for(M, iter = 3, E = 0)
REGE.FC(M, E = 1, iter = 3, until.change = TRUE, use.diag = TRUE, normE = FALSE)
REGE.FC.ow(M, E = 1, iter = 3, until.change = TRUE, use.diag = TRUE, normE = FALSE)
REGD.ne.for(M, iter = 3, E = 0)
REGD.ow.ne.for(M, iter = 3, E = 0)
REGE.ne.for(M, iter = 3, E = 1)
REGE.nm.diag.for(M, iter = 3, E = 1)
REGE.ownm.diag.for(M, iter = 3, E = 1)
REGE.ownm.ne.for(M, iter = 3, E = 1)

Arguments

M            Matrix or a 3 dimensional array representing the network. The third dimension allows for several relations to be analyzed.
E            Initial (dis)similarity in terms of regular equivalence.
itern        The desired number of iterations.
until.change  Should the iterations be stopped when no change occurs.
use.diag     Should the diagonal be used. If FALSE, all diagonal elements are set to 0.
normE        Should the equivalence matrix be normalized after each iteration.

Value

E            A matrix of (dis)similarities in terms of regular equivalence.
Eall         An array of (dis)similarity matrices in terms of regular equivalence, each third dimension represents one iteration. For "for" functions, only the initial and the final (dis)similarities are returned.
M            Matrix or a 3 dimensional array representing the network used in the call.
iter         The desired number of iterations.
use.diag     Should the diagonal be used - for functions implemented in R only.
...

Author(s)

Aleš Žiberna based on Douglas R. White's original REGE and REGD
Reordering an image matrix of the blockmodel (or an error matrix based on new and old partition)

Reorder an image matrix of the blockmodel (or an error matrix based on new and old partition. The partitions should be the same, except that classes can have different labels. It is useful when we want to have a different order of classes in figures and then also in image matrices. Currently it is only suitable for one-mode blockmodels.
Usage

reorderImage(IM, oldClu, newClu)

Arguments

IM An image or error matrix.
oldClu Old partition.
newClu New partition, the same as the old one except for class labels.

Value

Reorder matrix (rows and columns are reordered).

Author(s)

Ales Ziberna

References


See Also

critFunC, plot.mat, clu, IM, err

Description

The functions compute the distances in terms of Structural equivalence (Lorrain and White, 1971) between the units of a one-mode network. Several options for treating the diagonal values are supported.
Arguments

M
A matrix representing the (usually valued) network. For now, only one-relational networks are supported. The network must be one-mode.

method
The method used to compute distances - any of the methods allowed by functions dist, "cor" or "cov" (all package::stats) or just "cor" or "cov" (given as a character).

fun
Which function should be used to compute distances (given as a character).

fun.on.rows
For non-standard function - does the function compute measure on rows (such as "cor", "cov"...) of the data matrix (as opposed to computing measure on columns (such as dist).

handle.interaction
How should the interaction between the vertices analysed be handled:
"switch" (the default) - assumes that when comparing units i and j, M[i,i] should be compared with M[j,j] and M[i,j] with M[j,i]. These two comparisons are weighted by 2. This should be used with Euclidean distance to get the corrected Euclidean distance with p = 2.
"switch2" - the same (alias)
"switch1" - the same as above, only that the two comparisons are weighted by 1. This should be used with Euclidean distance to get the corrected Euclidean distance with p = 1.
"ignore" (diagonal) - Diagonal is ignored. This should be used with Euclidean distance to get the corrected Euclidean distance with p = 0.
"none" - the matrix is used "as is"

use
For use with methods "cor" and "cov", for other methods (the default option should be used if handle.interaction == "ignore"), "pairwise.complete.obs" are always used, if stats.dist.cor.cov = TRUE.

Details

If both method and fun are "default", the Euclidean distances are computed. The "default" method for fun = "dist" is "euclidean" and for fun = "cor" "pearson".

Value

A matrix (usually of class dist) is returned.

Author(s)

Aleš Žiberna

References


See Also
dist, hclust, REGE, optParC, optParC, optRandomParC

Examples

# Generating a simple network corresponding to the simple Sum of squares
# Structural equivalence with blockmodel:
# null com
# null null
n <- 20
net <- matrix(NA, ncol = n, nrow = n)
clu <- rep(1:2, times = c(5, 15))
tclu <- table(clu)
net[clu == 1, clu == 1] <- rnorm(n = tclu[1] * tclu[1], mean = 0, sd = 1)
net[clu == 1, clu == 2] <- rnorm(n = tclu[1] * tclu[2], mean = 4, sd = 1)
net[clu == 2, clu == 1] <- rnorm(n = tclu[2] * tclu[1], mean = 0, sd = 1)
net[clu == 2, clu == 2] <- rnorm(n = tclu[2] * tclu[2], mean = 0, sd = 1)

D <- sedist(M = net)
plot.mat(net, clu = cutree(hclust(d = D, method = "ward"), k = 2))

ss

Sum of Squared deviations from the mean and sum of Absolute Devia-
tions from the median

Description

Functions to compute Sum of Squared deviations from the mean and sum of Absolute Deviations from the median.

Usage

ss(x)
ad(x)

Arguments

x A numeric vector.

Value

Sum of Squared deviations from the mean or sum of Absolute Deviations from the median.

Author(s)

Aleš Žiberna
two2one

Two-mode network conversions

Description

Converting two mode networks from two to one mode matrix representation and vice versa. If a two-mode matrix is converted into a one-mode matrix, the original two-mode matrix lies in the upper right corner of the one-mode matrix.

Usage

two2one(M, clu = NULL)
one2two(M, clu = NULL)

Arguments

M        A matrix representing the (usually valued) network.
clu      A partition. Each unique value represents one cluster. This should be a list of two vectors, one for each mode.

Value

Function returns list with the elements: a two mode matrix of a the two mode network in its upper left corner.

M        The matrix.
clu      The partition, in form appropriate for the mode of the matrix.

Author(s)

Aleš Žiberna

See Also

optParC, optParC, optRandomParC, plot.mat

Examples

# Generating a simple network corresponding to the simple Sum of squares
# Structural equivalence with blockmodel:
# null com
# null null
n <- c(7, 13)
net <- matrix(NA, nrow = n[1], ncol = n[2])
clu <- list(rep(1:2, times = c(3, 4)), rep(1:2, times = c(5, 8)))
tclu <- lapply(clu, table)
net[clu[[1]] == 1, clu[[2]] == 1] <- rnorm(n = tclu[[1]][1] * tclu[[2]][1],
mean = 0, sd = 1)
net[clu[[1]] == 1, clu[[2]] == 2] <- rnorm(n = tclu[[1]][1] * tclu[[2]][2],
mean = 0, sd = 1)
net[clu[[1]] == 2, clu[[2]] == 1] <- rnorm(n = tclu[[1]][1] * tclu[[2]][1],
mean = 0, sd = 1)
net[clu[[1]] == 2, clu[[2]] == 2] <- rnorm(n = tclu[[1]][1] * tclu[[2]][2],
mean = 0, sd = 1)
net[clu[[1]] == 1, clu[[2]] == 1] <- rnorm(n = tclu[[1]][1] * tclu[[2]][1],
mean = 0, sd = 1)
net[clu[[1]] == 1, clu[[2]] == 2] <- rnorm(n = tclu[[1]][1] * tclu[[2]][2],
mean = 0, sd = 1)

mean = 4, sd = 1)
mean = 4, sd = 1)
mean = 0, sd = 1)
plot.mat(net, clu = clu) # Two mode matrix of a two mode network

# Converting to one mode network
M1 <- two2one(net)$M
plot.mat(M1, clu = two2one(net)$clu) # Plotting one mode matrix
# Converting one to two mode matrix and plotting
plot.mat(one2two(M1, clu = clu)$M, clu = clu)
Index

*Topic **algebra**
genMatrixMult, 15

*Topic **array**
genMatrixMult, 15

*Topic **character**
formatA, 13

*Topic **cluster**
blockmodeling-package, 2
critFunC, 5
find.m, 11
fun.by.blocks, 14
genRandomPar, 16
nkpartitions, 20
optRandomParC, 21
rand, 34
REGE, 35
sedist, 38
two2one, 41

*Topic **datasets**
notesBorrowing, 21

*Topic **file**
Pajek, 26

*Topic **graphs**
blockmodeling-package, 2
critFunC, 5
gplot1, 17
optRandomParC, 21
Pajek, 26
plot.mat, 27
REGE, 35
sedist, 38
two2one, 41

*Topic **hplot**
plot.mat, 27

*Topic **manip**
clu, 3
ircNorm, 19
recode, 34
reorderImage, 37

*Topic **math**
fun.by.blocks, 14

*Topic **package**
blockmodeling-package, 2

*Topic **univar**
ss, 40

ad (ss), 40

blockmodeling-package, 2
clu, 3, 9, 38
crand (rand), 34
crand2 (rand), 34
critFunC, 3, 4, 5, 13, 21, 24, 25, 27, 33, 37, 38
dist, 40

EM (clu), 3
eff, 9, 38
eff (clu), 3

find.cut, 13
find.cut (find.m), 11
find.m, 11, 13
find.m2, 13
find.m2 (find.m), 11
formatA, 13
fun.by.blocks, 14

genMatrixMult, 15
genRandomPar, 16, 23
gplot1, 17
gplot2 (gplot1), 17

hclust, 40

IM, 9, 38
IM (clu), 3
ircNorm, 19
loadmatrix (Pajek), 26
loadnetwork (Pajek), 26
loadnetwork2 (Pajek), 26
loadnetwork3 (Pajek), 26
loadnetwork4 (Pajek), 26
loadpajek (Pajek), 26
loadvector (Pajek), 26
loadvector2 (Pajek), 26

matmult, 16

network, 3
nkpar (nkpartitions), 20
nkpartitions, 20
notesBorrowing, 21

one2two (twozone), 41
optParC, 3, 13, 15, 37, 40, 41
optParC (critFunC), 5
optRandomParC, 3, 4, 9, 15–17, 21, 27, 33, 40, 41

Pajek, 26
partitions (clu), 3
plot, 18
plot.array (plot.mat), 27
plot.crit.fun, 9
plot.crit.fun (plot.mat), 27
plot.mat, 3, 4, 27, 27, 37, 38, 41
plot.opt.more.par (plot.mat), 27
plot.opt.par (plot.mat), 27
plotArray (plot.mat), 27
plotMat, 13
plotMat (plot.mat), 27
plotMatNm (plot.mat), 27

rand, 34
rand2 (rand), 34
recode, 34
REGD.for (REGE), 35
REGD.ne.for (REGE), 35
REGD.ow.for (REGE), 35
REGD.ow.ne.for (REGE), 35
REGE, 3, 35, 40
reorderImage, 37

savematrix (Pajek), 26
savenetwork (Pajek), 26
savevector (Pajek), 26
sedist, 37, 38
sna, 3