Package 'StochBlock'

July 25, 2025

Type Package

Title Stochastic Blockmodeling of One-Mode and Linked Networks

Version 0.1.5

Date 2025-07-23

Maintainer Aleš Žiberna <ales.ziberna@fdv.uni-lj.si>

Description Stochastic blockmodeling of one-mode and linked networks as presented in Škulj and Žiberna (2022) <doi:10.1016/j.socnet.2022.02.001>. The optimization is done via CEM (Classification Expectation Maximization) algorithm that can be initialized by random partitions or the results of k-means algorithm. The development of this package is financially supported by the Slovenian Research Agency (<https://www.arrs.si/>) within the research programs P5-0168 and the research projects J7-8279 (Blockmodeling multilevel and temporal networks) and J5-2557 (Comparison and evaluation of different approaches to blockmodeling dynamic networks by simulations with application to Slovenian co-authorship networks).

License GPL (>= 2)

Imports blockmodeling, doParallel, doRNG, foreach, Rcpp (>= 1.0.0)

LinkingTo Rcpp, RcppArmadillo

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation yes

Author Aleš Žiberna [aut, cre] (ORCID:
<https://orcid.org/0000-0003-1534-6971>),
Fabio Ashtar Telarico [ctb] (ORCID:
<https://orcid.org/0000-0002-8740-7078>)

Repository CRAN

Date/Publication 2025-07-25 07:10:07 UTC

Contents

`findActiveParam` *Finds the active model's parameters*

Description

Finds the active model's parameters

Usage

```
findActiveParam(M, n, k, na.rm = TRUE)
```

Arguments

`M`	matrix
`n`	number of units (equal to number of `M`'s rows)
`k`	parameters to retrieve
`na.rm`	logical, whether to ignore `NA` data

Value

An array containing the parameters

`ICLStochBlock` *Function that computes integrated classification likelihood based on stochastic one-mode and linked block modeling. If* `clu` *is a list, the method for linked/multilevel networks is applied. The support for multirelational networks is not tested.*

Description

Function that computes integrated classification likelihood based on stochastic one-mode and linked block modeling. If `clu` is a list, the method for linked/multilevel networks is applied. The support for multirelational networks is not tested.

Usage

```
ICLStochBlock(
  M,
  clu,
  weights = NULL,
  uWeights = NULL,
  diagonal = c("ignore", "seperate", "same"),
  limitType = c("none", "inside", "outside"),
  limits = NULL,
  weightClusterSize = 1,
  addOne = TRUE,
  eps = 0.001
)
```

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.
`clu`	A partition. Each unique value represents one cluster. If the nework 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.
`weights`	The weights for each cell in the matrix/array. A matrix or an array with the same dimmensions as `M`.
`uWeights`	The weights for each unin. A vector with the length equal to the number of units (in all sets).
`diagonal`	How should the diagonal values be treated. Possible values are: • ignore - diagonal values are ignored • seperate - diagonal values are treated seperately • same - diagonal values are treated the same as all other values
`limitType`	Type of limit to use. Forced to 'none' if `limits` is `NULL`. Otherwise, one of either `outer` or `inner`.
`limits`	If `diagonal` is `"ignore"` or `"same"`, an array with dimensions equal to: • number of clusters (of all types) • number of clusters (of all types) • number of relations • 2 - the first is lower limit and the second is upper limit If `diagonal` is `"seperate"`, a list of two array. The first should be as described above, representing limits for off diagonal values. The second should be similar with only 3 dimensions, as one of the first two must be omitted.
`weightClusterSize`	The weight given to cluster sizes (logprobabilites) compared to ties in loglikelihood. Defaults to 1, which is "classical" stochastic blockmodeling.

addOne	Should one tie with the value of the tie equal to the density of the superBlock be added to each block to prevent block means equal to 0 or 1 and also "shrink" the block means toward the superBlock mean. Defaults to TRUE.
eps	If addOne = FALSE, the minimal deviation from 0 or 1 that the block mean/density can take.

Value

The value of ICL

See Also

llStochBlock; weightsMlLoglik

Examples

```
# Create a synthetic network matrix
set.seed(2022)
library(blockmodeling)
k<-2 # number of blocks to generate
blockSizes<-rep(20,k)
IM<-matrix(c(0.8,.4,0.2,0.8), nrow=2)
clu<-rep(1:k, times=blockSizes)
n<-length(clu)
M<-matrix(rbinom(n*n,1,IM[clu,clu]),ncol=n, nrow=n)
clu<-sample(1:2,nrow(M),replace=TRUE)
plotMat(M,clu) # Have a look at this random partition
ICL_pre<-ICLStochBlock(M,clu) # Calculate its ICL
ICL_pre
res<-stochBlock(M,clu=clu) # Optimizing the partition
plot(res) # Have a look at the optimized partition
ICL_post<-res$ICL # Calculate its ICL
ICL_post
# We expect the ICL pre-optimisation to be smaller:
ICL_pre<ICL_post
```

---

llStochBlock *Function that computes criterion function used in stochastic one-mode and linked blockmodeling. If* clu *is a list, the method for linked/multilevel networks is applied*

Description

Function that computes criterion function used in stochastic one-mode and linked blockmodeling. If clu is a list, the method for linked/multilevel networks is applied

Usage

```
llStochBlock(
  M,
  clu,
  weights = NULL,
  uWeights = NULL,
  diagonal = c("ignore", "seperate", "same"),
  limitType = c("none", "inside", "outside"),
  limits = NULL,
  weightClusterSize = 1,
  addOne = TRUE,
  eps = 0.001
)
```

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.
`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.
`weights`	The weights for each cell in the matrix/array. A matrix or an array with the same dimensions as `M`.
`uWeights`	The weights for each unit. A vector with the length equal to the number of units (in all sets).
`diagonal`	How should the diagonal values be treated. Possible values are: • ignore - diagonal values are ignored • seperate - diagonal values are treated separately • same - diagonal values are treated the same as all other values
`limitType`	Type of limit to use. Forced to 'none' if `limits` is `NULL`. Otherwise, one of either `outer` or `inner`.
`limits`	If `diagonal` is `"ignore"` or `"same"`, an array with dimensions equal to: • number of clusters (of all types) • number of clusters (of all types) • number of relations • 2 - the first is lower limit and the second is upper limit If `diagonal` is `"seperate"`, a list of two array. The first should be as described above, representing limits for off diagonal values. The second should be similar with only 3 dimensions, as one of the first two must be omitted.
`weightClusterSize`	The weight given to cluster sizes (log-probabilities) compared to ties in loglikelihood. Defaults to 1, which is "classical" stochastic blockmodeling.

addOne — Should one tie with the value of the tie equal to the density of the superBlock be added to each block to prevent block means equal to 0 or 1 and also "shrink" the block means toward the superBlock mean. Defaults to TRUE.

eps — If addOne = FALSE, the minimal deviation from 0 or 1 that the block mean/density can take.

Value

- the value of the log-likelihood criterion for the partition `clu` on the network represented by `M` for binary stochastic blockmodel.

Author(s)

Aleš, Žiberna

References

Škulj, D., & Žiberna, A. (2022). Stochastic blockmodeling of linked networks. Social Networks, 70, 240-252, doi:10.1371/journal.pcbi.1005697.

Examples

```
# Create a synthetic network matrix
set.seed(2022)
library(blockmodeling)
k<-2 # number of blocks to generate
blockSizes<-rep(20,k)
IM<-matrix(c(0.8,.4,0.2,0.8), nrow=2)
clu<-rep(1:k, times=blockSizes)
n<-length(clu)
M<-matrix(rbinom(n*n,1,IM[clu,clu]),ncol=n, nrow=n)
clu<-sample(1:2,nrow(M),replace=TRUE)
plotMat(M,clu) # Have a look at this random partition
ll_pre<-llStochBlock(M,clu) # Calculate its loglikelihood
res<-stochBlockORP(M,k=2,rep=10) # Optimizing the partition
plot(res) # Have a look at the optimized partition
ll_post<-llStochBlock(M,clu(res)) # Calculate its loglikelihood
# We expect the loglikelihood pre-optimization to be smaller:
(-ll_pre)<(-ll_post)
```

`stochBlock` — *Function that performs stochastic one-mode and linked blockmodeling by optimizing a single partition. If* `clu` *is a list, the method for linked/multilevel networks is applied*

Description

Function that performs stochastic one-mode and linked blockmodeling by optimizing a single partition. If `clu` is a list, the method for linked/multilevel networks is applied

Usage

```
stochBlock(
  M,
  clu,
  weights = NULL,
  uWeights = NULL,
  diagonal = c("ignore", "seperate", "same"),
  limitType = c("none", "inside", "outside"),
  limits = NULL,
  weightClusterSize = 1,
  addOne = TRUE,
  eps = 0.001
)
```

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.
`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.
`weights`	The weights for each cell in the matrix/array. A matrix or an array with the same dimensions as `M`.
`uWeights`	The weights for each unin. A vector with the length equal to the number of units (in all sets).
`diagonal`	How should the diagonal values be treated. Possible values are: • ignore - diagonal values are ignored • seperate - diagonal values are treated seperately • same - diagonal values are treated the same as all other values
`limitType`	Type of limit to use. Forced to 'none' if `limits` is `NULL`. Otherwise, one of either `outer` or `inner`.
`limits`	If `diagonal` is `"ignore"` or `"same"`, an array with dimensions equal to: • number of clusters (of all types) • number of clusters (of all types) • number of relations • 2 - the first is lower limit and the second is upper limit If `diagonal` is `"seperate"`, a list of two array. The first should be as described above, representing limits for off diagonal values. The second should be similar with only 3 dimensions, as one of the first two must be omitted.

`weightClusterSize`
: The weight given to cluster sizes (logprobabilites) compared to ties in loglikelihood. Defaults to 1, which is "classical" stochastic blockmodeling.

`addOne`
: Should one tie with the value of the tie equal to the density of the superBlock be added to each block to prevent block means equal to 0 or 1 and also "shrink" the block means toward the superBlock mean. Defaults to TRUE.

`eps`
: If addOne = FALSE, the minimal deviation from 0 or 1 that the block mean/density can take.

Value

A list of class `opt.par` normally passed other commands with `StockBlockORP` and containing:

`clu`
: A vector (a list for multi-mode networks) indicating the cluster to which each unit belongs;

`IM`
: Image matrix of this partition;

`weights`
: The weights for each cell in the matrix/array. A matrix or an array with the same dimensions as `M`.

`uWeights`
: The weights for each unit. A vector with the length equal to the number of units (in all sets).

`err`
: - weighted log-likelihood.

`ICL`
: Integrated Criterion Likelihood for this partition

Author(s)

Aleš, Žiberna

References

Škulj, D., & Žiberna, A. (2022). Stochastic blockmodeling of linked networks. Social Networks, 70, 240-252, doi:10.1371/journal.pcbi.1005697.

See Also

`stochBlockORP`

Examples

```
# Create a synthetic network matrix
set.seed(2022)
library(blockmodeling)
k<-2 # number of blocks to generate
blockSizes<-rep(20,k)
IM<-matrix(c(0.8,.4,0.2,0.8), nrow=2)
clu<-rep(1:k, times=blockSizes)
n<-length(clu)
M<-matrix(rbinom(n*n,1,IM[clu,clu]),ncol=n, nrow=n)
clu<-sample(1:2,nrow(M),replace=TRUE)
plotMat(M,clu) # Have a look at this random partition
```

```
res<-stochBlock(M,clu) # Optimising the partition
plot(res) # Have a look at the optimised parition

# Create a synthetic linked-network matrix
set.seed(2022)
library(blockmodeling)
IM<-matrix(c(0.8,.4,0.2,0.8), nrow=2)
clu<-rep(1:2, each=20) # Partition to generate
n<-length(clu)
nClu<-length(unique(clu)) # Number of clusters to generate
M1<-matrix(rbinom(n^2,1,IM[clu,clu]),ncol=n, nrow=n) # First network
M2<-matrix(rbinom(n^2,1,IM[clu,clu]),ncol=n, nrow=n) # Second network
M12<-diag(n) # Linking network
nn<-c(n,n)
k<-c(2,2)
Ml<-matrix(0, nrow=sum(nn),ncol=sum(nn))
Ml[1:n,1:n]<-M1
Ml[n+1:n,n+1:n]<-M2
Ml[n+1:n, 1:n]<-M12
plotMat(Ml) # Linked network
clu1<-sample(1:2,nrow(M1),replace=TRUE)
clu2<-sample(3:4,nrow(M1),replace=TRUE)
plotMat(Ml,list(clu1,clu2)) # Have a look at this random partition
res<-stochBlock(Ml,list(clu1,clu2)) # Optimising the partition
plot(res) # Have a look at the optimised parition
```

`stochBlockKMint` *A function for using k-means to initialized the stochastic one-mode and linked blockmodeling.*

Description

A function for using k-means to initialized the stochastic one-mode and linked blockmodeling.

Usage

```
stochBlockKMint(
  M,
  k,
  nstart = 100,
  perm = 0,
  sharePerm = 0.2,
  save.initial.param = TRUE,
  deleteMs = TRUE,
  max.iden = 10,
  return.all = FALSE,
  return.err = TRUE,
  seed = NULL,
```

```
  maxTriesToFindNewPar = perm * 10,
  skip.par = NULL,
  printRep = ifelse(perm <= 10, 1, round(perm/10)),
  n = NULL,
  nCores = 1,
  useParLapply = FALSE,
  cl = NULL,
  stopcl = is.null(cl),
  ...
)
```

Arguments

M	A square matrix giving the adjaciency relationg between the network's nodes (aka vertexes)
k	The number of clusters used in the generation of partitions.
nstart	number of random starting points for the classical k-means algorithm (for each set of units). Defaults to 100.
perm	Number or partitions obtained by randomly permuting the k-means partition - if 0, no permutations are made, only the original partition is analyzed.
sharePerm	The probability that a unit will have their randomly assigned. Defaults to 0.20.
save.initial.param	Should the inital parameters(approaches, ...) of using stochBlock be saved. The default value is TRUE.
deleteMs	Delete networks/matrices from the results of to save space. Defaults to TRUE.
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).
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. Defaults to TRUE.
seed	Optional. The seed for random generation of partitions.
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.
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.
useParLapply	Should parLapplyLB be used (otherwise foreach is used). Defaults to true as it needs less dependencies. It might be removed in future releases and only allow the use of parLapplyLB.

cl	The cluster to use (if formed beforehand). Defaults to NULL.
stopcl	Should the cluster be stopped after the function finishes. Defaults to is.null(cl).
...	Arguments passed to other functions, see stochBlock.

Value

A list containing:

M	The one- or multi-mode 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 stochblock, only without M.
err	If return.err = TRUE - The vector of errors or inconsistencies of the empirical network with the ideal partitions.
nIter	The vector of the iterations on each starting partition. If many of the values equalmaxiter, then maxiter may be too small.
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 to this function.
initial.param	If selected - The initial parameters are used.

Author(s)

Aleš, Žiberna

References

Škulj, D., & Žiberna, A. (2022). Stochastic blockmodeling of linked networks. Social Networks, 70, 240-252.

---

stochBlockORP *A function for optimizing multiple random partitions using stochastic one-mode and linked blockmodeling. Calls stochBlock for optimizing individual partitions.*

Description

A function for optimizing multiple random partitions using stochastic one-mode and linked blockmodeling. Calls stochBlock for optimizing individual partitions.

Usage

```
stochBlockORP(
  M,
  k,
  rep,
  save.initial.param = TRUE,
  deleteMs = TRUE,
  max.iden = 10,
  return.all = FALSE,
  return.err = TRUE,
  seed = NULL,
  parGenFun = blockmodeling::genRandomPar,
  mingr = NULL,
  maxgr = NULL,
  addParam = list(genPajekPar = TRUE, probGenMech = NULL),
  maxTriesToFindNewPar = rep * 10,
  skip.par = NULL,
  printRep = ifelse(rep <= 10, 1, round(rep/10)),
  n = NULL,
  nCores = 1,
  useParLapply = FALSE,
  cl = NULL,
  stopcl = is.null(cl),
  ...
)
```

Arguments

M	A square matrix giving the adjaciency relationg between the network's nodes (aka vertexes)
k	The number of clusters used in the generation of partitions.
rep	The number of repetitions/different starting partitions to check.
save.initial.param	Should the inital parameters(approaches, ...) of using stochBlock be saved. The default value is TRUE.
deleteMs	Delete networks/matrices from the results of to save space. Defaults to TRUE.
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).
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. Defaults to TRUE.
seed	Optional. The seed for random generation of partitions.
parGenFun	The function (object) that will generate random partitions. The default function is genRandomPar. The function has to accept the following parameters: k (number o 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.
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.
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.
useParLapply	Should parLapplyLB be used (otherwise foreach is used). Defaults to true as it needs less dependencies. It might be removed in future releases and only allow the use of parLapplyLB.
cl	The cluster to use (if formed beforehand). Defaults to NULL.
stopcl	Should the cluster be stopped after the function finishes. Defaults to is.null(cl).
...	Arguments passed to other functions, see stochBlock.

Value

A list of class "opt.more.par" containing:

M	The one- or multi-mode 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 stochblock, only without M.
err	If return.err = TRUE - The vector of errors or inconsistencies = -log-likelihoods.
ICL	Integrated classification likelihood for the best partition.
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 to this function.
initial.param	If selected - The initial parameters are used.
Random.seed	.Random.seed at the end of the function.
cl	Cluster used for parallel computations if supplied as an input parameter.

Warning

It should be noted that the time needed to optimise the partition depends on the number of units (aka nodes) in the networks as well as the number of clusters due to the underlying algorithm. Hence, partitioning networks with 100 units and large number of blocks (e.g., >5) can take a long time (from 20 minutes to a few hours or even days).

Author(s)

Aleš, Žiberna

References

Škulj, D., & Žiberna, A. (2022). Stochastic blockmodeling of linked networks. Social Networks, 70, 240-252, doi:10.1371/journal.pcbi.1005697.

See Also

stochBlock

Examples

```
# Simple one-mode network
library(blockmodeling)
k<-2
blockSizes<-rep(20,k)
IM<-matrix(c(0.8,.4,0.2,0.8), nrow=2)
if(any(dim(IM)!=c(k,k))) stop("invalid dimensions")

set.seed(2021)
clu<-rep(1:k, times=blockSizes)
n<-length(clu)
M<-matrix(rbinom(n*n,1,IM[clu,clu]),ncol=n, nrow=n)
diag(M)<-0
plotMat(M)

resORP<-stochBlockORP(M,k=2, rep=10, return.all = TRUE)
resORP$ICL
plot(resORP)
clu(resORP)


# Linked network
library(blockmodeling)
set.seed(2021)
IM<-matrix(c(0.8,.4,0.2,0.8), nrow=2)
clu<-rep(1:2, each=20)
n<-length(clu)
nClu<-length(unique(clu))
M1<-matrix(rbinom(n^2,1,IM[clu,clu]),ncol=n, nrow=n)
M2<-matrix(rbinom(n^2,1,IM[clu,clu]),ncol=n, nrow=n)
M12<-diag(n)
nn<-c(n,n)
k<-c(2,2)
Ml<-matrix(0, nrow=sum(nn),ncol=sum(nn))
Ml[1:n,1:n]<-M1
Ml[n+1:n,n+1:n]<-M2
Ml[n+1:n, 1:n]<-M12
plotMat(Ml)
```

```
resMl<-stochBlockORP(M=Ml, k=k, n=nn, rep=10)
resMl$ICL
plot(resMl)
clu(resMl)
```

---

upAndDownSearch *Perform Up-and-Down Search Optimization*

Description

The upAndDownSearch function performs an up-and-down search (increasing and lowering the number of clusters) on a given dataset, starting from an initial partition and fitness value. The function assumes that the higher fitness value is better. Experimental! The stochBlockForUDS is just a wrapper to stochBlock function to be used within upAndDownSearch function.

Usage

```
upAndDownSearch(
  data,
  initPart,
  initFit,
  optimFun,
  nRep = 100,
  minPmove = 0.2,
  pFitSumTreshMoveBest = 5,
  ...
)

stochBlockForUDS(data, initPart, ...)
```

Arguments

data	A data object (e.g., data frame or matrix) on which the search is performed.
initPart	Initial partition or configuration to start the search from. It can be a vector for simple datasets and a list of vectors for temporal or linked networks
initFit	Initial fitness value associated with the initial partition.
optimFun	A function used for optimization. A function must accept a data and a initial partition and return a list with an element part holding the final partition and an element fit fitness value.
nRep	Integer. The number of repetitions for the search algorithm. Defaults to 100.
minPmove	Numeric. The minimum probability of moving to a new partition when the fitness does not improve. Defaults to 0.2.
pFitSumTreshMoveBest	Numeric. The threshold for the sum of 1 - probabilities of moving to a new partition before reverting to the best partition. Defaults to 5.
...	Additional paramters to optimFun.

Value

A list with the following components:

data The input data.

finalPart The final partition obtained.

finalFit The final fitness value after the search.

searchHistory A list containing the history of partitions and fitness values during the search.

callUsed The call used to invoke the function, capturing the parameters passed.

initial.param A list of initial parameters used in the function call withiut the data.

Examples

```
# Create a synthetic network matrix
set.seed(2022)
library(blockmodeling)
k<-2 # number of blocks to generate
blockSizes<-rep(20,k)
IM<-matrix(c(0.8,.4,0.2,0.8), nrow=2)
clu<-rep(1:k, times=blockSizes)
n<-length(clu)
M<-matrix(rbinom(n*n,1,IM[clu,clu]),ncol=n, nrow=n)
initClu<-rep(1, times=n)
initFit<-ICLStochBlock(M, initClu) # Initial fitness value
# Using up-and-down search to optimise the partition
res<-upAndDownSearch(data=M,initPart=initClu, initFit=initFit, optimFun=stochBlockForUDS, nRep=10)
plotMat(res$data, clu=res$bestPart) # Have a look at the optimised parition
print(res$bestFit) # Print the final fitness value

# Create a synthetic linked-network matrix
set.seed(2022)
library(blockmodeling)
IM<-matrix(c(0.9,.5,0.1,0.8), nrow=2)
clu<-rep(1:2, each=20) # Partition to generate
n<-length(clu)
nClu<-length(unique(clu)) # Number of clusters to generate
M1<-matrix(rbinom(n^2,1,IM[clu,clu]),ncol=n, nrow=n) # First network
M2<-matrix(rbinom(n^2,1,IM[clu,clu]),ncol=n, nrow=n) # Second network
M12<-diag(n) # Linking network
nn<-c(n,n)
k<-c(2,2)
Ml<-matrix(0, nrow=sum(nn),ncol=sum(nn))
Ml[1:n,1:n]<-M1
Ml[n+1:n,n+1:n]<-M2
Ml[n+1:n, 1:n]<-M12
plotMat(Ml) # Linked network
clu1<-rep(1, n)
clu2<-rep(2, n)
initClu<-list(clu1, clu2)
initFit<-ICLStochBlock(Ml, initClu) # Initial fitness value
# Using up-and-down search to optimise the partition
```

```
res<-upAndDownSearch(data=Ml,initPart=initClu, initFit=initFit, optimFun=stochBlockForUDS, nRep=10)
plotMat(res$data, clu=res$bestPart) # Have a look at the optimised parition
print(res$bestFit) # Print the final fitness value
```

`weightsMlLoglik` *Computes weights for parts of the multilevel network based on random errors using the SS approach with complete blocks only (compatible with k-means)*

Description

Computes weights for parts of the multilevel network based on random errors using the SS approach with complete blocks only (compatible with k-means)

Usage

```
weightsMlLoglik(
  mlNet,
  cluParts,
  k,
  mWeights = 1000,
  sumFun = sd,
  nCores = 0,
  weightClusterSize = 0,
  paramGenPar = list(genPajekPar = FALSE),
  ...
)
```

Arguments

`mlNet`	A multilevel/linked network - The code assumes only one relation –> a matrix.
`cluParts`	A partition spliting the units into different sets
`k`	A vecotor of number of clusters for each set of units in the network.
`mWeights`	The number of repetitions for computing random errors. Defaults to 1000
`sumFun`	The function to compute the summary of errors, which is then used to compute the weights by computing 1/summary. Defaults to `sd`.
`nCores`	The number of to use for parallel computing. 0 means all available - 1, 1 means only once core - no parallel computing.
`weightClusterSize`	The weight given to cluster sizes. Defalults to 0, as only this is weighted my the tie-based weights.
`paramGenPar`	The parameter `addParam` from `genRandomPar` (see documentation there). Default here is paramGenPar=list(genPajekPar = FALSE), which is different from the default in `genRandomPar`. The same value is used for generating partitions for all partitions.
`...`	Paramters passed to `llStochBlock`

Value

Weights and "intermediate results":

`errArr`	A 3d array of errors (`mWeights` for each part of the network)
`errMatSum`	`errArr` summed over all repetitions.
`weightsMat`	A matrix of weights, one for each part. An inverse of `errMatSum` with NaNs replaced by zeros.

Author(s)

Aleš, Žiberna

References

Škulj, D., & Žiberna, A. (2022). Stochastic blockmodeling of linked networks. Social Networks, 70, 240-252.

See Also

`llStochBlock`; `ICLStochBlock`

Index