Package 'TGS'

November 16, 2018

Title Rapid Reconstruction of Time-Varying Gene Regulatory Networks

Version 1.0.0

Description Rapid advancement in high-throughput gene expression measurement technologies has resulted in genome- scale time series datasets. Uncovering the underlying temporal sequence of gene regulatory events in the form of time-varying Gene Regulatory Networks (GRNs) demands computationally fast, accurate and highly scalable algorithms. To provide a flexible framework in a significantly time-efficient manner, a novel algorithm, namely TGS (Ashish Anand et al., 2018 <doi:10.1109/TCBB.2018.2861698>), is proposed here. TGS is shown to consume only 29 minutes for a microarray dataset with 4028 genes. Moreover, it provides the flexibility and time-efficiency, without losing the accuracy. Nevertheless, TGS's main memory requirement grows exponentially with the number of genes, which it tackles by restricting the maximum number of regulators for each gene. Relaxing this restriction remains an important challenge as the true number of regulators is not known a prior.

License CC BY-NC-SA 4.0

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

biocViews

Imports minet, stats, base, utils, bnstruct, ggm, foreach, doParallel, rjson

Suggests R.rsp

VignetteBuilder R.rsp

NeedsCompilation no

Author Manan Gupta [aut, cre],
Saptarshi Pyne [aut],
Alok Kumar [aut],
Ashish Anand [aut]

Maintainer Manan Gupta <guptamanan100@gmail.com>

Repository CRAN

Date/Publication 2018-11-16 16:10:13 UTC

R topics documented:

`adjmxToSif` *Create a .sif file from given adjacency matrix*

Description

Given a network adjacency matrix, creates an equivalent '.sif' file that is readable in Cytoscape. SIF file extension stands for Simple Interaction Format. Ref: http://manual.cytoscape.org/en/3.4.0/Supported_Network_File_For format E.g., if the input network adjacency matrix is as follows: (rows = src nodes, cols = tgt nodes, (A, B) = 0 and 1 implies that the edge 'A->B' does not exist and does exist, resp.) src\ tgt A B C D A 0 1 1 0 B 0 0 0 0 C 1 0 0 0 C 0 0 0 0 then the output SIF file will contain the following four lines: A B C B C A D The way Cytoscape interprets each line in this SIF file is as follows: <Source node name> <Target node1 name (if any)> <Target node2 name (if any)> ... Note that the values are delimited by tab.

Usage

```
adjmxToSif(adj.mx, output.dirname = "./OUTPUT")
```

Arguments

adj.mx	adjacency matrix that needs to be converted
output.dirname	name of the output directory where the file will be stored

---

calcPerfDiNet *Calculating performance metrics of the directed net 'inferredNet' w.r.t. the directed net 'targetNet'.*

Description

Calculating performance metrics of the directed net 'inferredNet' w.r.t. the directed net 'targetNet'.

Usage

```
calcPerfDiNet(inferredNet, targetNet, Result, n)
```

Arguments

inferredNet	directed net 'inferredNet'
targetNet	directed net 'targetNet'
Result	Matrix to store the results
n	Number of nodes in each of the nets.

Value

the performance metrics

Examples

```
res<-calcPerfDiNet(matrix(c(1,0,1,1,1,1,1,0,1),nrow=3,ncol=3),
+ matrix(c(0,1,0,1,1,1,1,0,1),nrow=3,ncol=3),
+ matrix(nrow=1,ncol=11),3)
```

checkUnrolledDbn *Checks whether the given unrolled DBN follows 1st Markov order or not*

Description

Checks whether the given unrolled DBN follows 1st Markov order or not

Usage

```
checkUnrolledDbn(unrolled.DBN.adj.matrix)
```

Arguments

unrolled.DBN.adj.matrix	An unrolled DBN adjacency matrix

Value

whether the given unrolled DBN follows 1st Markov order or not

Examples

```
checkUnrolledDbn(matrix(c(0,0,0,0),nrow=2,ncol=2))
```

CompareNet *Checks if 'di.net.adj.matrix' = 'cmi.net.adj.matrix'*

Description

Checks if 'di.net.adj.matrix' = 'cmi.net.adj.matrix'

Usage

```
CompareNet(di.net.adj.matrix, cmi.net.adj.matrix, num.node)
```

Arguments

di.net.adj.matrix	First adjacency metric
cmi.net.adj.matrix	Second adjacency metric
num.node	Number of nodes in the metrices

Value

Returns 1 if 'di.net.adj.matrix' = 'cmi.net.adj.matrix' else returns 0

Examples

```
CompareNet(matrix(c(0,0,0,0),nrow=2,ncol=2),
+ matrix(c(0,0,0,0),nrow=2,ncol=2),
+ 2)

CompareNet(matrix(c(0,0,1,0),nrow=2,ncol=2),
+ matrix(c(0,0,0,0),nrow=2,ncol=2),
+ 2)
```

`computeCmi` *Compute Conditional Mutual Infortion (CMI)*

Description

Compute Conditional Mutual Info between random variables 'v1' and 'v2' given random vector 'vcs'.

Usage

```
computeCmi(v1, v2, vcs)
```

Arguments

`v1`	random variable 1
`v2`	random variable 2
`vcs`	random vector 'vcs'

Value

Conditional Mutual info between 'v1' and 'v2'

Examples

```
computeCmi(c(3,5),c(3,5))
```

`ComputeCmiPcaCmi` *Compute Conditional Mutual Information (CMI) the way it is done in the implementation of the PCA-CMI algo*

Description

Compute Conditional Mutual Info between random variables 'v1' and 'v2' given random vector 'vcs'. Therefore, when 'vcs' is NULL, this function returns the mutual info between random variables 'v1' and 'v2'. The implementation of this R function is adapted from the mutual info estimator function used in the PCA-CMI algo [1]. The original funciton, namely 'cmi()' [2], is written in MATLAB.

Usage

```
ComputeCmiPcaCmi(v1, v2, vcs)
```

Arguments

`v1`	random variable 1
`v2`	random variable 2
`vcs`	random vector 'vcs'

Value

Conditional Mutual info between 'v1' and 'v2'

References

[1] Zhang X, Zhao X M, He K, et al. Inferring gene regulatory networks from gene expression data by path consistency algorithm based on conditional mutual information[J]. Bioinformatics, 2012, 28(1): 98-104. Companion website: https://sites.google.com/site/xiujunzhangcsb/software/pca-cmi

[2] Function declaration: 'function cmiv=cmi(v1,v2,vcs)', Source code file: http://www.compsysbio.org/grn/pca_cmi.m .

Examples

```
ComputeCmiPcaCmi(c(3,5),c(4,2))
```

ComputEntropy *Compute Entropy matrix from the input data*

Description

Compute Entropy matrix from the input data

Usage

```
ComputEntropy(input.data)
```

Arguments

input.data	input data matrix

Value

entropy matrix

Examples

```
df = data.frame(c(2,3,5), c(1,3,2), c(2,31,4))
ComputEntropy(df)
```

ConvertDinetToUndinet *Given a directed network, convert it into an undirected network*

Description

Given a directed network, convert it into an undirected network

Usage

```
ConvertDinetToUndinet(di.net)
```

Arguments

di.net	a directed network

Value

directed network converted to undirected network

Examples

```
ConvertDinetToUndinet(matrix(c(0,1,0,0),nrow=2))
```

CountFeedFwdEdgesUndi *Count the number of feed-forward edges in a given undirected network.*

Description

Count the number of feed-forward edges in a given undirected network.

Usage

```
CountFeedFwdEdgesUndi(undi.net.adj.matrix)
```

Arguments

undi.net.adj.matrix	adjacency matrix of an undirected network

Value

count of the number of feed-forward edges

Examples

```
CountFeedFwdEdgesUndi(matrix(c(0,1,0,1,0,1,0,0,0),nrow=3))
```

discretizeData.2L.Tesla *Discretize input data into 2 levels.*

Description

Discretize input data into the following two levels as done in For each gene, the expression values are first sorted; then the top 2 extreme values in either end of the sorted list are discarded; last, the median of the remaining values is used as the threshold above which the value is binarized as 'Level 2' and 'Level 1' otherwise. Here, 2 means the expression of a gene is up-regulated, and 1 means down-regulated.

Usage

```
discretizeData.2L.Tesla(input.data)
```

Arguments

input.data	data frame to be discretized. rows have samples and cols have variables.

Value

data discretized into 2 levels

References

1. Ahmed, Amr, and Eric P. Xing. "Recovering time-varying networks of dependencies in social and biological studies." Proceedings of the National Academy of Sciences 106.29 (2009): 11878-11883.]

---

discretizeData.2L.wt.l

Discretizes input data into two levels.

Description

Discretizes input data into the following two levels: Less than wild type => level 1 Greater than equal to wild type => level 2

Usage

```
discretizeData.2L.wt.l(input.data, input.wt.data.filename)
```

Arguments

input.data	the data that is given as the input, data frame to be discretized
input.wt.data.filename	path of the file containing the table

Value

input data discretized into 2 levels

---

discretizeData.2L.wt.le

Discretizes input data into two levels.

Description

Discretizes input data into the following two levels: Less than wild type => level 1 Greater than equal to wild type => level 2

Usage

```
discretizeData.2L.wt.le(input.data, input.wt.data.filename)
```

Arguments

input.data	the data that is given as the input, data frame to be discretized
input.wt.data.filename	path of the file containing the table

Value

input data discretized into 2 levels

---

discretizeData.3L.wt *Discretizes input data into three levels, given a tolerance.*

Description

Discretizes input data into three levels, given a tolerance. Let, expression value of gene i in a particular sample is x_i and wild type expression value of i is wt(i). Level 1: 0 =< x_i < (wt(i) - tolerance). Gene i is down regulated or knocked out. Level 2: (wt(i) - tolerance) =< x_i =< (wt(i) + tolerance). Expression of gene i is at a steady-state. Level 3: (wt(i) + tolerance) < x_i =< 1. Gene i is up regulated.

Usage

```
discretizeData.3L.wt(input.data, input.wt.data.filename, tolerance,
  num.discr.levels)
```

Arguments

input.data	the data that is given as the input, data frame to be discretized
input.wt.data.filename	path of the file containing the table
tolerance	the tolerance
num.discr.levels	number of discrete levels

Value

input data discretized into 3 levels

discretizeData.5L.wt *Discretizes input data into five levels.*

Description

Discretizes input data into the following five levels: Let, expression value of gene i in a particular sample is x_i and wild type expression value of i is wt(i). Level 1: x_i = 0. Gene i is knocked out. Level 2: 0 < x_i < wt(i). Gene i is down regulated but not knocked out. Level 3: x_i = wt(i). Expression of gene i is at a steady-state. Level 4: wt(i) < x_i < 1. Gene i is up regulated but not maximally activated. Level 5: x_i = 1. Gene i is maximally activated.

Usage

```
discretizeData.5L.wt(input.data, input.wt.data.filename, num.discr.levels)
```

Arguments

`input.data`	the data that is given as the input, data frame to be discretized
`input.wt.data.filename`	path of the file containing the table
`num.discr.levels`	number of discrete levels

Value

input data discretized into 5 levels

eval.wrt.known.gene.ias *Accuracy of predicted directed gene reuglatory network adjacency matrix*

Description

Given a predicted directed gene reuglatory network adjacency matrix, evaluate its accuracy w.r.t. known gene-gene interactions.

Usage

```
eval.wrt.known.gene.ias(di.net.adj.matrix)
```

Arguments

`di.net.adj.matrix`	predicted directed gene reuglatory network adjacency matrix

References

[1] Song, Le, Mladen Kolar, and Eric P. Xing. "KELLER: estimating time-varying interactions between genes." Bioinformatics 25.12 (2009): i128-i136.

`GenTrueAdjMatrix` *Generates True net adjacency matrix and save as an R object*

Description

Generates True net adjacency matrix and save as an R object

Usage

```
GenTrueAdjMatrix(input.file, output.file, num.nodes)
```

Arguments

`input.file`	full path of the input file containing adjacency list
`output.file`	full path where output adjacency matrix is to be stored
`num.nodes`	the number of nodes

Value

also returns the adjacency matrix

`LearnClr2NetMfi` *Learns CLR2 network*

Description

Learns CLR2 net. For each node, retains top 'max.fanin' number of neighbours w.r.t. edge weight and removes rest of the edges. Tie is broken in favour of the neighbour having smaller node index. If there are less than that number of edges for a node, then retain all its neighbours.

Usage

```
LearnClr2NetMfi(input.data.discr, num.nodes, node.names, num.timepts,
  max.fanin, output.dirname = "./OUTPUT", mi.net.adj.matrix)
```

Arguments

input.data.discr	Discretized dataset
num.nodes	number of nodes
node.names	name of nodes
num.timepts	number of timepoints
max.fanin	number of top neighbours to retain
output.dirname	output directory name
mi.net.adj.matrix	mi network adjacency matrix

Value

mi network adjacency matrix

LearnClr3NetMfi *Learn CLR3 network*

Description

Learns CLR3 net. For each node, retains top 'max.fanin' number of neighbours w.r.t. edge weight and removes rest of the edges. Tie is broken in favour of the neighbour having smaller node index. If there are less than that number of edges for a node, then retain all its neighbours.

Usage

```
LearnClr3NetMfi(input.data.discr.3D, num.nodes, node.names, num.timepts,
  max.fanin, mi.net.adj.matrix.list)
```

Arguments

input.data.discr.3D	3D Discretized dataset
num.nodes	number of nodes
node.names	name of nodes
num.timepts	number of timepoints
max.fanin	number of top neighbours to retain
mi.net.adj.matrix.list	mi network adjacency matrix list

Value

mi network adjacency matrix list

LearnClrNetFromDiscrData *Learns CLR network from a given discretized dataset.*

Description

Learns CLR net from a given discretized dataset. For each node, retains top 'max.fanin' number of neighbours w.r.t. edge weight and removes rest of the edges. Tie is broken in favour of the neighbour having smaller node index. If there are less than that number of edges for a node, then retain all its neighbours.

Usage

```
LearnClrNetFromDiscrData(input.data.discr, num.nodes, node.names,
  num.timepts, max.fanin, output.dirname = "./OUTPUT")
```

Arguments

input.data.discr	Discretized dataset
num.nodes	number of nodes
node.names	name of nodes
num.timepts	number of timepoints
max.fanin	number of top neighbours to retain
output.dirname	output directory to store files

Value

mi network adjacency matrix

LearnClrNetMfi *Learns CLR network*

Description

Learns CLR net. For each node, retains top 'max.fanin' number of neighbours w.r.t. edge weight and removes rest of the edges. Tie is broken in favour of the neighbour having smaller node index. If there are less than that number of edges for a node, then retain all its neighbours.

Usage

```
LearnClrNetMfi(mut.info.matrix, mi.net.adj.matrix, num.nodes, max.fanin,
  output.dirname = "./OUTPUT")
```

Arguments

mut.info.matrix	matrix containing mut info
mi.net.adj.matrix	mi network adjacency matrix
num.nodes	number of nodes
max.fanin	number of top neighbours to retain
output.dirname	output directory to store files

Value

mi network adjacency matrix

LearnClrNetMfiVer2.1 *Learn CLR2.1 network*

Description

Learns CLR2.1 net. For each node, retains top 'max.fanin' number of neighbours w.r.t. edge weight and removes rest of the edges. Tie is broken in favour of the neighbour having smaller node index. If there are less than that number of edges for a node, then retain all its neighbours.

Usage

```
LearnClrNetMfiVer2.1(input.data.discr, num.nodes, node.names, num.timepts,
  max.fanin, output.dirname = "./OUTPUT", mi.net.adj.matrix)
```

Arguments

input.data.discr	Discretized dataset
num.nodes	number of nodes
node.names	name of nodes
num.timepts	number of timepoints
max.fanin	number of top neighbours to retain
output.dirname	output directory name
mi.net.adj.matrix	mi network adjacency matrix

Value

mi network adjacency matrix

learnCmiNetStruct *Learns the CMI structure*

Description

Learns the CMI structure

Usage

```
learnCmiNetStruct(di.net.adj.matrix, input.data, num.nodes, beta)
```

Arguments

di.net.adj.matrix	a directed network adjacency matrix
input.data	data given as input data frame
num.nodes	the number of nodes
beta	value of parameter beta

Value

the directed network adjacency matrix after learning the CMI network

learnDbnStructLayer3dParDeg1 *Unrolled DBN structure learning with Markov Order 0 and 1.*

Description

Unrolled DBN structure learning with Markov Order 0 and 1. Candidate parents: The target node itself and its CLR net neighbours at immediately previous and current time pt.

Usage

```
learnDbnStructLayer3dParDeg1(input.data.discr.3D, mi.net.adj.matrix,
  num.discr.levels, num.nodes, num.timepts, output.dirname = "./OUTPUT")
```

Arguments

`input.data.discr.3D`
: Dimensions 1 = time points, 2 = variables, 3 = samples under the same time point.

`mi.net.adj.matrix`
: Adjacency matrix of the mutual information network. Rownames and colnames should be node names.

`num.discr.levels`
: If input data is discretized, then number of discrete levels for each variable. Else if input data is continuous, then number of levels in which data needs to be discretized for performing the DBN structure learning.

`num.nodes`
: number of nodes

`num.timepts`
: number of timepoints

`output.dirname`
: output directory to store files

Value

Unrolled DBN adjacency matrix

`LearnDbnStructMo1Clr3Ser` *Learns DBN structure of Markov order 1 where candidate parents are selected using the CLR3 algo.*

Description

Learns DBN structure of Markov order 1 where candidate parents are selected using the CLR3 algo. It is a serial algorithmic implementation.

Usage

```
LearnDbnStructMo1Clr3Ser(input.data.discr.3D,
  mi.net.adj.matrix.list.filename, num.discr.levels, num.nodes,
  num.timepts, max.fanin, node.names, unrolled.DBN.adj.matrix.list)
```

Arguments

`input.data.discr.3D`
: Dimensions 1 = time points, 2 = variables, 3 = samples under the same time point.

`mi.net.adj.matrix.list.filename`
: List of filenames (relative/absolute path) of adjacency matrices of the mutual information network.

num.discr.levels
: If input data is discretized, then number of discrete levels for each variable. Else if input data is continuous, then number of levels in which data needs to be discretized for performing the DBN structure learning.

num.nodes
: number of nodes

num.timepts
: number of timepoints

max.fanin
: maximum incoming edges in the graph

node.names
: name of the nodes

unrolled.DBN.adj.matrix.list
: List of unrolled DBN adjacency matrices

Value

Unrolled DBN adjacency matrix

---

learnDbnStructMo1Layer3dParDeg1

Goal: Unrolled DBN structure learning with Markov Order 1.

Description

Goal: Unrolled DBN structure learning with Markov Order 1. Candidate parents: The target node itself and its CLR net neighbours at immediately previous time pt.

Usage

```
learnDbnStructMo1Layer3dParDeg1(input.data.discr.3D, mi.net.adj.matrix,
  num.discr.levels, num.nodes, num.timepts, max.fanin, output.dirname)
```

Arguments

input.data.discr.3D
: Dimensions 1 = time points, 2 = variables, 3 = samples under the same time point.

mi.net.adj.matrix
: Adjacency matrix of the mutual information network. Rownames and colnames should be node names.

num.discr.levels
: If input data is discretized, then number of discrete levels for each variable. Else if input data is continuous, then number of levels in which data needs to be discretized for performing the DBN structure learning.

num.nodes
: number of nodes

num.timepts
: number of timepoints

max.fanin
: maximum incoming edges in the graph

output.dirname
: output directory to store files

Value

Unrolled DBN adjacency matrix

`learnDbnStructMo1Layer3dParDeg1_v2`
Goal: Unrolled DBN structure learning with Markov Order 1.

Description

This function is the newer version of learnDbnStructMo1Layer3dParDeg1(). The only difference is in the size of unrolled DBN adjacency matrix. In earlier version, the size is ((V \ times T) \ times (v \ times T)) where V = number of nodes and T = number of time points. But the size is too large when V is very large. For e.g., when V = 4028, the function can not execute successfully even with 32 GB main memory in grni server. In this version, the size is reduced by storing the unrolled DBN in an adjacency list of length (T - 1). The t^th element in the list is the predicted network adjacency matrix at the t^th time interval. Therefore, each list element is a binary matrix of dimension (V \ times V). Hence, the total size of the unrolled DBN adjacency list is ((T - 1) \ times (V \ times V)). Candidate parents: The target node itself and its CLR net neighbours at immediately previous time pt.

Usage

```
learnDbnStructMo1Layer3dParDeg1_v2(input.data.discr.3D, mi.net.adj.matrix,
  num.discr.levels, num.nodes, num.timepts, max.fanin, node.names,
  clr.algo)
```

Arguments

`input.data.discr.3D`	Dimensions 1 = time points, 2 = variables, 3 = samples under the same time point.
`mi.net.adj.matrix`	Adjacency matrix of the mutual information network. Rownames and colnames should be node names.
`num.discr.levels`	If input data is discretized, then number of discrete levels for each variable. Else if input data is continuous, then number of levels in which data needs to be discretized for performing the DBN structure learning.
`num.nodes`	number of nodes
`num.timepts`	number of timepoints
`max.fanin`	maximum incoming edges in the graph
`node.names`	name of the nodes
`clr.algo`	clr algorithm to use

Value

Unrolled DBN adjacency matrix

LearnMiNetStructClr *Learns the CLR network Replaces all non-zero edge weights with 1.*

Description

Learns the CLR network Replaces all non-zero edge weights with 1.

Usage

```
LearnMiNetStructClr(mut.info.matrix, mi.net.adj.matrix, num.nodes,
  output.dirname = "./OUTPUT")
```

Arguments

mut.info.matrix	matrix containing mut info
mi.net.adj.matrix	mi network adjacency matrix
num.nodes	number of nodes
output.dirname	output directory to store files

Value

mi network adjacency matrix

LearnMiNetStructRowMedian *Learn the mi network structure*

Description

Learn the mi network structure

Usage

```
LearnMiNetStructRowMedian(mut.info.matrix, mi.net.adj.matrix, num.nodes)
```

Arguments

mut.info.matrix	matrix containing mut info
mi.net.adj.matrix	mi network adjacency matrix
num.nodes	number of nodes

Value

mi network adjacency matrix

Examples

```
LearnMiNetStructRowMedian(
+ matrix(c(0.1,0.5,0.53,0.76,0,0.12,0.43,0.65,0.23),nrow=3),
+ matrix(c(1,0,1,0,0,0,1,0,1),nrow=3),
+ 3)
```

LearnMiNetStructZstat *Learn the mi network structure*

Description

Learn the mi network structure

Usage

```
LearnMiNetStructZstat(mut.info.matrix, mi.net.adj.matrix, entropy.matrix,
  alpha)
```

Arguments

`mut.info.matrix`	matrix containing mut info
`mi.net.adj.matrix`	mi network adjacency matrix
`entropy.matrix`	matrix containing the entropy information
`alpha`	parameter value alpha

Value

mi network adjacency matrix

LearnTgs *Implementing the TGS Algorithm.*

Description

It Uncovers the underlying temporal sequence of gene regulatory events in the form of time-varying Gene Regulatory Networks (GRNs) It learns the time-varying GRN structures independently of each other, without imposing any structural constraint. However, it is time intensive and hence not suitable for large-scale GRNs.

Usage

```
LearnTgs(isfile = 0, input.data.filename = "", num.timepts = 0,
  true.net.filename = "", input.wt.data.filename = "",
  is.discrete = TRUE, num.discr.levels = 2, discr.algo = "",
  mi.estimator = "mi.pca.cmi", apply.aracne = FALSE,
  clr.algo = "CLR", max.fanin = 14, allow.self.loop = TRUE,
  scoring.func = "BIC", input.dirname = "", output.dirname = "",
  json.file = "")
```

Arguments

isfile	1 if the parameters are given in a json file otherwise 0.
input.data.filename	name of the file containing the data without the directory name. It can be .tsv or .Rdata file only.
num.timepts	number of distinct timepoints
true.net.filename	File containing the true network without the directory name. In case non empty then should contain .Rdata file with object name 'true.net.adj.matrix'
input.wt.data.filename	File containing Input Wild Type data without the directory name. If it is not empty hen must be a .tsv file having first row containing names of genes except (1,1) and second row should have the WT values except (2,1)th cell.
is.discrete	whether the data is discretized or not
num.discr.levels	number of discreteized levels that each gene has (if already discretized) or it should have (if it is to be discretized)
discr.algo	The algo to follow in case of not discretized. Possible values:- discretizeData.2L.Tesla,discretizeData.2L.v
mi.estimator	which method to use for estimating the mutual information matrix. Generally 'mi.pca.cmi' is used
apply.aracne	ARACNE is applied to refine the mutual information matrix. In case true then TGS+ version is used otherwise the original TGS variant is executed
clr.algo	CLR algo to use Possible values :- CLR, CLR2, CLR2.1, CLR3, spearman

max.fanin	the maximum number of regulators each gene can have
allow.self.loop	Whether to allow self loops in the graph
scoring.func	Which scoring func to use
input.dirname	Name of the directory where input files are. By default your current directory unless specified otherwise
output.dirname	Name of the directory where output files are to be stored. By default your current directory unless specified otherwise
json.file	name of the json file along with directory if parameters are to be read from json file

Examples

```
## Not run:
LearnTgs(0,input.data.filename = "DmLc3E.RData", num.timepts = 6, is.discrete = TRUE,
 num.discr.levels = 2, mi.estimator = "mi.pca.cmi", apply.aracne = FALSE,
 clr.algo = "CLR", max.fanin = 14, allow.self.loop = TRUE,
 input.dirname = "location where file is stored",
 output.dirname = "location where output needs to be stored")

LearnTgs(0,input.data.filename = "DmLc3L.RData", num.timepts = 2, is.discrete = TRUE,
 num.discr.levels = 2, mi.estimator = "mi.pca.cmi", apply.aracne = FALSE,
 clr.algo = "CLR", max.fanin = 14, allow.self.loop = TRUE,
 input.dirname = "location where file is stored",
 output.dirname = "location where output needs to be stored")

## End(Not run)
```

Print.common.di.edges *Given two di network adjacency matrices, it prints the common edges in an output file*

Description

Given two di nets' adjacency matrices (must of of same dim, same rownames, same colnames where rows = source nodes, cols = tgt nodes, 0 and 1 represent absence and presence of an edge, resp.), this function prints the common edges in an output text file.

Usage

```
Print.common.di.edges(di.net.adj.matrix1, di.net.adj.matrix2,
  output.dirname = "./OUTPUT")
```

Arguments

`di.net.adj.matrix1`	first di network adjacency matrix
`di.net.adj.matrix2`	second di network adjacency matrix
`output.dirname`	output directory to store files

`reachable.nodes` *Returns all the nodes reachable from the given node in the directed adjacency matrix*

Description

Given a directed network adjacency matrix and a node name, returns names of all the nodes reachable from the given node. In the given directed network adjacency matrix, rows = source nodes, cols = tgt nodes. (i, j)th cell = 1 implies a directed edge from i to j; = 0 implies no directed edge from i to j. Row names of the given matrix must be the names of the nodes.

Usage

```
reachable.nodes(di.net.adj.matrix, src.node.name)
```

Arguments

`di.net.adj.matrix`	The directed network adjacency matrix
`src.node.name`	The node whose reachable nodes need to be returned

Value

A vector containing all the nodes reachable from the given node

Examples

```
x = matrix(c(1,0,0,0,1,1,0,1,1),nrow=3)
rownames(x) <- c('A','B','C')
colnames(x) <- c('A','B','C')
reachable.nodes(x,'A')
```

rollDbn *Convert a given unrolled Dynamic Bayesian Network (DBN) into a rolled DBN using different rolling methods Rolls time-varying networks into a single time-invariant network. This function is compatible with the time-varying networks learnt through learnDbnStruct3dParDeg1.R::learnDbnStructMo1Layer3dParDeg1().*

Description

Convert a given unrolled Dynamic Bayesian Network (DBN) into a rolled DBN using different rolling methods

Rolls time-varying networks into a single time-invariant network. This function is compatible with the time-varying networks learnt through learnDbnStruct3dParDeg1.R::learnDbnStructMo1Layer3dParDeg1().

Usage

```
rollDbn(num.nodes, node.names, num.timepts, unrolled.DBN.adj.matrix,
  roll.method, allow.self.loop)
```

Arguments

num.nodes	Number of the desired nodes in the rolled DBN
node.names	Names of the desired nodes in the rolled DBN
num.timepts	Number of time points in the unrolled DBN
unrolled.DBN.adj.matrix	Given unrolled DBN adjacency matrix. It is a 2D matrix of dimension ((num.nodes X num.timepts) X (num.nodes X num.timepts)).
roll.method	Which rolling method to use from 'any', 'all', or some real number in (0, 1), like - 0.5.
allow.self.loop	Boolean to decide whether to allow self loop or not in the rolled DBN

Value

rolled.DBN.adj.matrix Return the rolled DBN adjacency matrix. It is a 2D matrix of dimension (num.nodes * num.nodes).

`rollDbn_v2` *Convert a given unrolled Dynamic Bayesian Network (DBN) into a rolled DBN using different rolling methods Rolls time-varying networks into a single time-invariant network. This function is compatible with the time-varying networks learnt through learnDbnStruct3dParDeg1.R::learnDbnStructMo1Layer3dParDeg1_v2().*

Description

Convert a given unrolled Dynamic Bayesian Network (DBN) into a rolled DBN using different rolling methods

Rolls time-varying networks into a single time-invariant network. This function is compatible with the time-varying networks learnt through learnDbnStruct3dParDeg1.R::learnDbnStructMo1Layer3dParDeg1_v2().

Usage

```
rollDbn_v2(num.nodes, node.names, num.timepts,
  unrolled.DBN.adj.matrix.list, roll.method, allow.self.loop)
```

Arguments

`num.nodes`	Number of the desired nodes in the rolled DBN
`node.names`	Names of the desired nodes in the rolled DBN
`num.timepts`	Number of time points in the unrolled DBN
`unrolled.DBN.adj.matrix.list`	Given time-varying network adjacency list. Its length = num.time.trans = (num.timepts - 1). The t^th element of the list represents the predicted network adjacency matrix of the t^th time transition. This matrix is of dimension (num.nodes \ times num.nodes).
`roll.method`	Which rolling method to use from 'any', 'all', or some real number in (0, 1), like - 0.5.
`allow.self.loop`	Boolean to decide whether to allow self loop or not in the rolled DBN

Value

rolled.DBN.adj.matrix Return the rolled DBN adjacency matrix. It is a 2D matrix of dimension (num.nodes * num.nodes).

Index