Package ‘SEMID’

March 26, 2026

Type Package

Title Identifiability of Linear Structural Equation Models

Version 0.5.1

Date 2026-03-26

Maintainer Nils Sturma <nils.sturma@epfl.ch>

Description Provides routines to check identifiability of linear structural equation models and factor analysis models. The routines are based on the graphical representation of structural equation models.

License GPL (>= 2)

Encoding UTF-8

URL https://github.com/Lucaweihs/SEMID

BugReports https://github.com/Lucaweihs/SEMID/issues

Imports R.oo (>= 1.20.0), R.methodsS3, igraph (>= 1.0.1), R.utils (>= 2.3.0), rje

Suggests testthat

NeedsCompilation no

Repository CRAN

RoxygenNote 7.3.3

Author Rina Foygel Barber [aut],
Mathias Drton [aut],
Miriam Kranzlmüller [aut],
Nils Sturma [cre, aut],
Luca Weihs [aut]

Date/Publication 2026-03-26 12:10:02 UTC

Contents

SEMID-package *SEMID package documentation.*

Description

SEMID provides a number of methods for testing the global/generic identifiability of mixed graphs and latent-factor graphs.

Details

The only functions you're likely to need from **SEMID** are semID and lfhtcID. A complete description of all package features, along with examples, can be found at https://github.com/Lucaweihs/SEMID.

Examples

```
###
# Checking the generic identifiability of parameters in a mixed graph.
###

# Mixed graphs are specified by their directed adjacency matrix L and
# bidirected adjacency matrix O.
L = t(matrix(
 c(0, 1, 1, 0, 0,
   0, 0, 1, 1, 1,
   0, 0, 0, 1, 0,
   0, 0, 0, 0, 1,
   0, 0, 0, 0, 0), 5, 5))

O = t(matrix(
 c(0, 0, 0, 1, 0,
   0, 0, 1, 0, 1,
   0, 0, 0, 0, 0,
   0, 0, 0, 0, 0,
   0, 0, 0, 0, 0), 5, 5)); O=O+t(O)

# Create a mixed graph object
graph = MixedGraph(L, O)

# We can plot what this mixed graph looks like, blue edges are directed
# red edges are bidirected.
plot(graph)

# Without using decomposition techniques we can't identify all nodes
# just using the half-trek criterion
htcID(graph, tianDecompose = FALSE)

# The edgewiseTSID function can show that all edges are generically
# identifiable without proprocessing with decomposition techniques
edgewiseTSID(graph, tianDecompose = FALSE)
```

```
# The above shows that all edges in the graph are generically identifiable.
# See the help of edgewiseTSID to find out more information about what
# else is returned by edgewiseTSID.

###
# Checking generic parameter identifiability using the generalGenericID
# function
###

L = t(matrix(
 c(0, 1, 0, 0, 0,
   0, 0, 0, 1, 1,
   0, 0, 0, 1, 0,
   0, 1, 0, 0, 1,
   0, 0, 0, 1, 0), 5, 5))

O = t(matrix(
 c(0, 0, 0, 0, 0,
   0, 0, 1, 0, 1,
   0, 0, 0, 1, 0,
   0, 0, 0, 0, 0,
   0, 0, 0, 0, 0), 5, 5)); O=O+t(O)

# Create a mixed graph object
graph = MixedGraph(L, O)

# Now lets define an "identification step" function corresponding to
# using the edgewise identification algorithm but with subsets
# controlled by 1.
restrictedEdgewiseIdentifyStep <- function(mixedGraph,
                                            unsolvedParents,
                                            solvedParents,
                                            identifier) {
     return(edgewiseIdentifyStep(mixedGraph, unsolvedParents,
                                 solvedParents, identifier,
                                 subsetSizeControl = 1))
}

# Now we run an identification algorithm that iterates between the
# htc and the "restricted" edgewise identification algorithm
generalGenericID(graph, list(htcIdentifyStep,
                               restrictedEdgewiseIdentifyStep),
                 tianDecompose = FALSE)

# We can do better (fewer unsolved parents) if we don't restrict the edgewise
# identifier algorithm as much
generalGenericID(graph, list(htcIdentifyStep, edgewiseIdentifyStep),
                  tianDecompose = FALSE)

###
# Checking the generic identifiability of parameters in a latent-factor graph.
###
```

```
# Latent digraphs are specified by their directed adjacency matrix L
library(SEMID)
L = matrix(c(0, 1, 0, 0, 0, 0,
             0, 0, 1, 0, 0, 0,
             0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 1, 0,
             0, 0, 0, 0, 0, 0,
             1, 1, 1, 1, 1, 0), 6, 6, byrow=TRUE)
observedNodes = seq(1,5)
latentNodes = c(6)

# Create the latent digraph object corresponding to L
g = LatentDigraph(L, observedNodes, latentNodes)

# Plot latent digraph
plot(g)

# We can identify all nodes by the latent-factor half-trek criterion
lfhtcID(g)
```

`ancestors` *All ancestors of a collection of nodes*

Description

Finds all the ancestors of a collection of nodes. These ancestors DO include the nodes themselves (every node is considered an ancestor of itself).

Usage

```
ancestors(this, nodes, ...)

## S3 method for class 'LatentDigraphFixedOrder'
ancestors(this, nodes, includeObserved = T, includeLatents = T, ...)

## S3 method for class 'LatentDigraph'
ancestors(this, nodes, includeObserved = T, includeLatents = T, ...)

## S3 method for class 'MixedGraph'
ancestors(this, nodes, ...)
```

Arguments

`this`	the graph object
`nodes`	the nodes from which to find all ancestors
`...`	ignored.

includeObserved
: if TRUE includes observed nodes in the returned set.

includeLatents
: if TRUE includes latent nodes in the returned set.

Value

the ancestors of the nodes in the observed part of the graph.

---

ancestralID — *Determines which edges in a mixed graph are ancestralID-identifiable*

Description

Uses the an identification criterion of Drton and Weihs (2015); this version of the algorithm is somewhat different from Drton and Weihs (2015) in that it also works on cyclic graphs. The original version of the algorithm can be found in the function `graphID.ancestralID`.

Usage

```
ancestralID(mixedGraph, tianDecompose = T)
```

Arguments

`mixedGraph`	a `MixedGraph` object representing the L-SEM.
`tianDecompose`	TRUE or FALSE determining whether or not the Tian decomposition should be used before running the current generic identification algorithm. In general letting this be TRUE will make the algorithm faster and more powerful.

Value

see the return of `generalGenericID`.

---

ancestralIdentifyStep — *Perform one iteration of ancestral identification.*

Description

A function that does one step through all the nodes in a mixed graph and tries to determine if directed edge coefficients are generically identifiable by leveraging decomposition by ancestral subsets. See Algorithm 1 of Drton and Weihs (2015); this version of the algorithm is somewhat different from Drton and Weihs (2015) in that it also works on cyclic graphs.

Usage

```
ancestralIdentifyStep(mixedGraph, unsolvedParents, solvedParents, identifier)
```

Arguments

`mixedGraph` a MixedGraph object representing the mixed graph.

`unsolvedParents` a list whose ith index is a vector of all the parents j of i in G which for which the edge j->i is not yet known to be generically identifiable.

`solvedParents` the complement of `unsolvedParents`, a list whose ith index is a vector of all parents j of i for which the edge i->j is known to be generically identifiable (perhaps by other algorithms).

`identifier` an identification function that must produce the identifications corresponding to those in solved parents. That is `identifier` should be a function taking a single argument Sigma (any generically generated covariance matrix corresponding to the mixed graph) and returns a list with two named arguments

Lambda denote the number of nodes in `mixedGraph` as n. Then Lambda is an nxn matrix whose i,jth entry

1. equals 0 if i is not a parent of j,
2. equals NA if i is a parent of j but `identifier` cannot identify it generically,
3. equals the (generically) unique value corresponding to the weight along the edge i->j that was used to produce Sigma.

Omega just as Lambda but for the bidirected edges in the mixed graph

such that if j is in `solvedParents[[i]]` we must have that Lambda[j,i] is not NA.

Value

a list with four components:

`identifiedEdges` a matrix rx2 matrix where r is the number of edges that where identified by this function call and `identifiedEdges[i,1]` -> `identifiedEdges[i,2]` was the ith edge identified

`unsolvedParents` as the input argument but updated with any newly identified edges

`solvedParents` as the input argument but updated with any newly identified edges

`identifier` as the input argument but updated with any newly identified edges

References

Drton, M. and Weihs, L. (2015) Generic Identifiability of Linear Structural Equation Models by Ancestor Decomposition. arXiv 1504.02992

bidirectedComponents *Get bidirected components of a mixed graph*

Description

Returns induced subgraphs of connected bidirected components with more than 1 node.

Usage

```
bidirectedComponents(graph)
```

Arguments

graph	a MixedGraph object representing the mixed graph.

Value

list, where each object is a MixedGraph with at least two nodes.

checkLocalBBCriterion *Check Local BB-Criterion*

Description

checks whether a graph is recursively identifiable by only using the local BB-criterion

Usage

```
checkLocalBBCriterion(adjMatrix, latentNodes, observedNodes)
```

Arguments

adjMatrix	of the graph
latentNodes	set of latent nodes
observedNodes	set of observed nodes

Value

a list consisting of a Boolean, whether the graph is sign-identifiable and if yes, a list consisting of the sets

References

Sturma, N., Kranzlmüller, M., Portakal, I., and Drton, M. (2025) Matching Criterion for Identifiability in Sparse Factor Analysis. arXiv:2502.02986

`checkMatchingCriterion` *Check Matching Criterion*

Description

for a latent node h, find a single observed node v and two sets of observed nodes W and U so that the tuple satisfies the matching criterion

Usage

```
checkMatchingCriterion(
  flowGraphAdjMatrix,
  adjMatrix,
  h,
  latentNodes,
  observedNodes,
  maxCard = length(observedNodes)
)
```

Arguments

`flowGraphAdjMatrix`	matrix representing the basis flow graph for any latent node h
`adjMatrix`	of the graph
`h`	latent node for which a tuple satisfying the criterion is sought
`latentNodes`	set of latent nodes
`observedNodes`	set of observed nodes
`maxCard`	maximum size of set W

Value

a list consisting of a Boolean, whether the graph is sign-identifiable and if yes, a list consisting of the sets

References

Sturma, N., Kranzlmüller, M., Portakal, I., and Drton, M. (2025) Matching Criterion for Identifiability in Sparse Factor Analysis. arXiv:2502.02986

children — *All children of a collection of nodes.*

Description

Returns all children of the collection (does not necessarily include the input nodes themselves unless they are parents of one another).

Usage

```
children(this, nodes, ...)

## S3 method for class 'LatentDigraphFixedOrder'
children(this, nodes, includeObserved = T, includeLatents = T, ...)

## S3 method for class 'LatentDigraph'
children(this, nodes, includeObserved = T, includeLatents = T, ...)

## S3 method for class 'MixedGraph'
children(this, nodes, ...)
```

Arguments

`this`	the graph object.
`nodes`	nodes the nodes of which to find the children
`...`	ignored.
`includeObserved`	if TRUE includes observed nodes in the returned set.
`includeLatents`	if TRUE includes latent nodes in the returned set.

Value

the observed children

createAncestralIdentifier — *Create an ancestral identification function.*

Description

A helper function for `ancestralIdentifyStep`, creates an identifier function based on its given parameters. This created identifier function will identify the directed edges from 'targets' to 'node.'

Usage

```
createAncestralIdentifier(
  idFunc,
  sources,
  targets,
  node,
  htrSources,
  ancestralSubset,
  cComponent
)
```

Arguments

`idFunc`	identification of edge coefficients often requires that other edge coefficients already be identified. This argument should be a function that produces all such identifications. The newly created identifier function will return these identifications along with its own.
`sources`	the sources of the half-trek system.
`targets`	the targets of the half-trek system (these should be the parents of `node`).
`node`	the node for which all incoming edges are to be identified (the tails of which are targets).
`htrSources`	the nodes in sources which are half-trek reachable from `node`. All incoming edges to these sources should be identified by `idFunc` for the newly created identification function to work.
`ancestralSubset`	an ancestral subset of the graph containing `node`.
`cComponent`	a list corresponding to the connected component containing `node` in the subgraph induced by `ancestralSubset`. See `tianDecompose` for how such connected component lists are formed.

Value

an identification function

`createEdgewiseIdentifier`

Create an edgewise identification function

Description

A helper function for `edgewiseIdentifyStep`, creates an identifier function based on its given parameters. This created identifier function will identify the directed edges from 'targets' to 'node.'

Usage

```
createEdgewiseIdentifier(
  idFunc,
  sources,
  targets,
  node,
  solvedNodeParents,
  sourceParentsToRemove
)
```

Arguments

`idFunc`	identification of edge coefficients often requires that other edge coefficients already be identified. This argument should be a function that produces all such identifications. The newly created identifier function will return these identifications along with its own.
`sources`	the sources of the half-trek system.
`targets`	the targets of the half-trek system (these should be the parents of `node`).
`node`	the node for which all incoming edges are to be identified (the tails of which are targets).
`solvedNodeParents`	the parents of node that have been solved
`sourceParentsToRemove`	a list of the parents of the sources that should have their edge to their respect source removed.

Value

an identification function

`createHtcIdentifier` *Create an htc identification function.*

Description

A helper function for `htcIdentifyStep`, creates an identifier function based on its given parameters. This created identifier function will identify the directed edges from 'targets' to 'node.'

Usage

```
createHtcIdentifier(idFunc, sources, targets, node, htrSources)
```

Arguments

- `idFunc` — identification of edge coefficients often requires that other edge coefficients already be identified. This argument should be a function that produces all such identifications. The newly created identifier function will return these identifications along with its own.
- `sources` — the sources of the half-trek system.
- `targets` — the targets of the half-trek system (these should be the parents of `node`).
- `node` — the node for which all incoming edges are to be identified (the tails of which are targets).
- `htrSources` — the nodes in sources which are half-trek reachable from `node`. All incoming edges to these sources should be identified by `idFunc` for the newly created identification function to work.

Value

an identification function

References

Foygel, R., Draisma, J., and Drton, M. (2012) Half-trek criterion for generic identifiability of linear structural equation models. *Ann. Statist.* 40(3): 1682-1713.

`createIdentifierBaseCase` — *Create an identifier base case*

Description

Identifiers are functions that take as input a covariance matrix Sigma corresponding to some mixed graph `G` and, from that covariance matrix, identify some subset of the coefficients in the mixed graph `G`. This function takes as input the matrices, `L` and `O`, defining `G` and creates an identifier that does not identify any of the coefficients of `G`. This is useful as a base case when building more complex identification functions.

Usage

```
createIdentifierBaseCase(L, O)
```

Arguments

- `L` — Adjacency matrix for the directed part of the path diagram/mixed graph; an edge pointing from i to j is encoded as L[i,j]=1 and the lack of an edge between i and j is encoded as L[i,j]=0. There should be no directed self loops, i.e. no i such that L[i,i]=1.
- `O` — Adjacency matrix for the bidirected part of the path diagram/mixed graph. Edges are encoded as for the L parameter. Again there should be no self loops. Also this matrix will be coerced to be symmetric so it is only necessary to specify an edge once, i.e. if O[i,j]=1 you may, but are not required to, also have O[j,i]=1.

Value

a function that takes as input a covariance matrix compatible with the mixed graph defined by `L/O` and returns a list with two named components:

`Lambda` a matrix equal to `L` but with `NA` values instead of `1`s

`Omega` a matrix equal to `O` but with `NA` values instead of `1`s

When building more complex identifiers these NAs will be replaced by the value that can be identified from Sigma.

`createLFHtcIdentifier` *Create a latent-factor half-trek critierion identification function.*

Description

A helper function for `lfhtcIdentifyStep`, creates an identifier function based on its given parameters. This created identifier function will identify the directed edges from 'targets' to 'node.'

Usage

```
createLFHtcIdentifier(idFunc, v, Y, Z, parents, reachableY)
```

Arguments

`idFunc`	identification of edge coefficients often requires that other edge coefficients already be identified. This argument should be a function that produces all such identifications. The newly created identifier function will return these identifications along with its own.
`v`	the node for which all incoming edges are to be identified (the tails of which are targets).
`Y`	the sources of the latent-factor half-trek system.
`Z`	the nodes that are reached from Y via an latent-factor half-trek of the form `y <- h -> z` where `h` is an element of L.
`parents`	the parents of node v.
`reachableY`	the nodes in Y which are latent-factor half-trek reachable from Z or v by avoiding the nodes in L. All incoming edges to these nodes should be identified by idFunc the newly created identification function to work.

Value

an identification function

References

Barber, R. F., Drton, M., Sturma, N., and Weihs L. (2022). Half-Trek Criterion for Identifiability of Latent Variable Models. *arXiv preprint* arXiv:2201.04457

createLFIdentifierBaseCase

Create an latent identifier base case

Description

Identifiers are functions that take as input a covariance matrix Sigma corresponding to some latent digraph G and, from that covariance matrix, identify some subset of the coefficients coresponding to the direct causal effects in the latent digraph G. This function takes as input the digraph G and creates an identifier that does not identify any of the direct causal effects. This is useful as a base case when building more complex identification functions.

Usage

```
createLFIdentifierBaseCase(graph)
```

Arguments

graph	a LatentDigraph object representing the latent-factor graph. All latent nodes in this graph should be source nodes (i.e. have no parents).

Value

a function that takes as input a covariance matrix compatible with the latent digraph defined by L and returns a list with two named components:

Lambda a matrix equal to the observed part of graph$L() but with NA values instead of 1s

Omega a matrix equal to graph$O() but with NA values for coefficients not equal to zero.

When building more complex identifiers these NAs will be replaced by the value that can be identified from the covariance matrix corresponding to G.

createSimpleBiDirIdentifier

Identify bidirected edges if all directed edges are identified

Description

Creates an identifier function that assumes that all directed edges have already been identified and then is able to identify all bidirected edges simultaneously.

Usage

```
createSimpleBiDirIdentifier(idFunc)
```

Arguments

`idFunc`	an identifier function that identifies all directed edges

Value

a new identifier function that identifies everything.

`createTrekFlowGraph` *Helper function to create a flow graph.*

Description

Helper function to create a flow graph.

Usage

```
createTrekFlowGraph(this, ...)

## S3 method for class 'LatentDigraphFixedOrder'
createTrekFlowGraph(this, ...)
```

Arguments

`this`	the graph object
`...`	ignored

`createTrekSeparationIdentifier` *Create an trek separation identification function*

Description

A helper function for `trekSeparationIdentifyStep`, creates an identifier function based on its given parameters. This created identifier function will identify the directed edge from 'parent' to 'node.'

Usage

```
createTrekSeparationIdentifier(
  idFunc,
  sources,
  targets,
  node,
  parent,
  solvedParents
)
```

Arguments

idFunc	identification of edge coefficients often requires that other edge coefficients already be identified. This argument should be a function that produces all such identifications. The newly created identifier function will return these identifications along with its own.
sources	the sources of the half-trek system.
targets	the targets of the half-trek system (these should be the parents of node).
node	the node for which all incoming edges are to be identified (the tails of which are targets).
parent	the parent of node for which the edge node -> parent should be generically identified.
solvedParents	the parents of node that have been solved

Value

an identification function

---

createTrGraph *Helper function to create a graph encoding trek reachable relationships.*

Description

Helper function to create a graph encoding trek reachable relationships.

Usage

```
createTrGraph(this, ...)

## S3 method for class 'LatentDigraphFixedOrder'
createTrGraph(this, ...)
```

Arguments

this	the graph object
...	ignored

descendants — *Get descendants of a collection of observed nodes*

Description

Finds all descendants of a collection of nodes, this DOES include the nodes themselves (every node is considered a descendant of itself).

Usage

```
descendants(this, nodes, ...)

## S3 method for class 'LatentDigraphFixedOrder'
descendants(this, nodes, includeObserved = T, includeLatents = T, ...)

## S3 method for class 'LatentDigraph'
descendants(this, nodes, includeObserved = T, includeLatents = T, ...)

## S3 method for class 'MixedGraph'
descendants(this, nodes, ...)
```

Arguments

`this`	the graph object
`nodes`	the nodes from which to get the descendants.
`...`	ignored.
`includeObserved`	if TRUE includes observed nodes in the returned set.
`includeLatents`	if TRUE includes latent nodes in the returned set.

edgewiseID — *Determines which edges in a mixed graph are edgewiseID-identifiable*

Description

Uses the edgewise identification criterion of Weihs, Robeva, Robinson, et al. (2017) to determine which edges in a mixed graph are generically identifiable.

Usage

```
edgewiseID(mixedGraph, tianDecompose = T, subsetSizeControl = 3)
```

Arguments

mixedGraph
: a MixedGraph object representing the L-SEM.

tianDecompose
: TRUE or FALSE determining whether or not the Tian decomposition should be used before running the current generic identification algorithm. In general letting this be TRUE will make the algorithm faster and more powerful.

subsetSizeControl
: a positive integer (Inf allowed) which controls the size of edgesets searched in the edgewiseID algorithm. Suppose, for example, this has value 3. Then if a node i has n parents, this will restrict the algorithm to only look at subsets of the parents of size 1,2,3 and n-2, n-1, n. Making this parameter smaller means the algorithm will be faster but less exhaustive (and hence less powerful).

Value

see the return of generalGenericID.

---

edgewiseIdentifyStep *Perform one iteration of edgewise identification.*

Description

A function that does one step through all the nodes in a mixed graph and tries to identify new edge coefficients using the existence of half-trek systems as described in Weihs, Robeva, Robinson, et al. (2017).

Usage

```
edgewiseIdentifyStep(
  mixedGraph,
  unsolvedParents,
  solvedParents,
  identifier,
  subsetSizeControl = Inf
)
```

Arguments

mixedGraph
: a MixedGraph object representing the mixed graph.

unsolvedParents
: a list whose ith index is a vector of all the parents j of i in G which for which the edge j->i is not yet known to be generically identifiable.

solvedParents
: the complement of unsolvedParents, a list whose ith index is a vector of all parents j of i for which the edge i->j is known to be generically identifiable (perhaps by other algorithms).

`identifier` an identification function that must produce the identifications corresponding to those in solved parents. That is `identifier` should be a function taking a single argument Sigma (any generically generated covariance matrix corresponding to the mixed graph) and returns a list with two named arguments

Lambda denote the number of nodes in `mixedGraph` as n. Then Lambda is an nxn matrix whose i,jth entry
1. equals 0 if i is not a parent of j,
2. equals NA if i is a parent of j but `identifier` cannot identify it generically,
3. equals the (generically) unique value corresponding to the weight along the edge i->j that was used to produce Sigma.

Omega just as Lambda but for the bidirected edges in the mixed graph

such that if j is in `solvedParents[[i]]` we must have that Lambda[j,i] is not NA.

`subsetSizeControl` a positive integer (Inf allowed) which controls the size of edgesets searched in the edgewiseID algorithm. Suppose, for example, this has value 3. Then if a node i has n parents, this will restrict the algorithm to only look at subsets of the parents of size 1,2,3 and n-2, n-1, n. Making this parameter smaller means the algorithm will be faster but less exhaustive (and hence less powerful).

Value

see the return of `htcIdentifyStep`.

`edgewiseTSID` *Determines which edges in a mixed graph are edgewiseID+TS identifiable*

Description

Uses the edgewise+TS identification criterion of Weihs, Robeva, Robinson, et al. (2017) to determine which edges in a mixed graph are generically identifiable. In particular this algorithm iterates between the half-trek, edgewise, and trek-separation identification algorithms in an attempt to identify as many edges as possible, this may be very slow.

Usage

```
edgewiseTSID(
  mixedGraph,
  tianDecompose = T,
  subsetSizeControl = 3,
  maxSubsetSize = 3
)
```

Arguments

`mixedGraph`	a `MixedGraph` object representing the L-SEM.
`tianDecompose`	TRUE or FALSE determining whether or not the Tian decomposition should be used before running the current generic identification algorithm. In general letting this be TRUE will make the algorithm faster and more powerful.
`subsetSizeControl`	a positive integer (Inf allowed) which controls the size of edgesets searched in the edgewiseID algorithm. Suppose, for example, this has value 3. Then if a node i has n parents, this will restrict the algorithm to only look at subsets of the parents of size 1,2,3 and n-2, n-1, n. Making this parameter smaller means the algorithm will be faster but less exhaustive (and hence less powerful).
`maxSubsetSize`	a positive integer which controls the maximum subset size considered in the trek-separation identification algorithm. Making this parameter smaller means the algorithm will be faster but less exhaustive (and hence less powerful).

Value

see the return of `generalGenericID`.

---

extmID *Check Extended M-Identifiability*

Description

Check Extended M-Identifiability

Usage

```
extmID(lambda, maxCard = length(observedNodes))
```

Arguments

`lambda`	adjacency matrix with number of cols = number of latent nodes, number of rows = number of observed nodes
`maxCard`	(optional) maximum size of set W

Value

a list consisting of a Boolean, whether the graph is sign-identifiable and if yes, a list consisting of the sets

References

Sturma, N., Kranzlmüller, M., Portakal, I., and Drton, M. (2025) Matching Criterion for Identifiability in Sparse Factor Analysis. arXiv:2502.02986l

findColumnsWithSumOne *A Helper Function for Check ZUTA*

Description

iterative function that tries to find a column with sum=1, delete the row with the 1 in that column, check smaller matrix

Usage

```
findColumnsWithSumOne(cleanMatrix)
```

Arguments

`cleanMatrix`	a matrix without zero columns and rows

Value

a Boolean whether it finds a column with sum=1 and the new matrix, without the row in which the entry is, fulfills ZUTA

flowBetween *Flow from one set of nodes to another.*

Description

Flow from one set of nodes to another.

Usage

```
flowBetween(this, sources, sinks)

## S3 method for class 'FlowGraph'
flowBetween(this, sources, sinks)
```

Arguments

`this`	the flow graph object
`sources`	the nodes from which flow should start.
`sinks`	the nodes at which the flow should end.

Value

a list with two named components, `value` (the size of the computed flow) and `activeSources` (a vector representing the subset of sources which have non-zero flow out of them for the found max-flow).

FlowGraph *Construct FlowGraph object*

Description

Creates an object representing a flow graph.

Usage

```
FlowGraph(L = matrix(0,1,1), vertexCaps = 1, edgeCaps = matrix(1,1,1))
```

Arguments

`L`	the adjacency matrix for the flow graph. The (i,j)th of L should be a 1 if there is an edge from i to j and 0 otherwise.
`vertexCaps`	the capacity of the vertices in the flow graph, should either be a single number or a vector whose ith entry is the capacity of vertex i.
`edgeCaps`	the capacities of the edges in the the flow graph, should be a matrix of the same dimensions as L with (i,j)th entry the capacity of the i->j edge.

Value

An object representing the FlowGraph.

generalGenericID *A general generic identification algorithm template.*

Description

A function that encapsulates the general structure of our algorithms for testing generic identifiability. Allows for various identification algorithms to be used in concert, in particular it will use the identifier functions in the list `idStepFunctions` sequentially until it can find no more identifications. The step functions that are currently available for use in `idStepFunctions` are

1. htcIdentifyStep,
2. ancestralIdentifyStep,
3. edgewiseIdentifyStep,
4. trekSeparationIdentifyStep.

Usage

```
generalGenericID(mixedGraph, idStepFunctions, tianDecompose = T)
```

Arguments

mixedGraph	a MixedGraph object representing the L-SEM.
idStepFunctions	a list of identification step functions
tianDecompose	TRUE or FALSE determining whether or not the Tian decomposition should be used before running the current generic identification algorithm. In general letting this be TRUE will make the algorithm faster and more powerful.

Value

returns an object of class 'GenericIDResult,' this object is just a list with 9 components:

`solvedParents` a list whose ith element contains a vector containing the subsets of parents of node i for which the edge j->i could be shown to be generically identifiable.

`unsolvedParents` as for `solvedParents` but for the unsolved parents.

`solvedSiblings` as for `solvedParents` but for the siblings of node i (i.e. the bidirected neighbors of i).

`unsolvedSiblings` as for `solvedSilbings` but for the unsolved siblings of node i (i.e. the bidirected neighbors of i).

`identifier` a function that takes a (generic) covariance matrix corresponding to the graph and identifies the edges parameters from solvedParents and solvedSiblings. See `htcIdentifyStep` for a more in-depth discussion of identifier functions.

`mixedGraph` a mixed graph object of the graph.

`idStepFunctions` a list of functions used to generically identify parameters. For instance, htcID uses the function `htcIdentifyStep` to identify edges.

`tianDecompose` the argument tianDecompose.

`call` the call made to this function.

---

getAncestors — *Get getAncestors of nodes in a graph.*

Description

Get the getAncestors of a collection of nodes in a graph g, the getAncestors DO include the the nodes themselves.

Usage

```
getAncestors(g, nodes)
```

Arguments

g	the graph (as an igraph).
nodes	the nodes in the graph of which to get the getAncestors.

Value

a sorted vector of all ancestor nodes.

---

getDescendants *Get descendants of nodes in a graph.*

Description

Gets the descendants of a collection of nodes in a graph (all nodes that can be reached by following directed edges from those nodes). Descendants DO include the nodes themselves.

Usage

```
getDescendants(g, nodes)
```

Arguments

`g`	the graph (as an igraph).
`nodes`	the nodes in the graph of which to get the descendants.

Value

a sorted vector of all descendants of nodes.

---

getHalfTrekSystem *Determines if a half-trek system exists in the mixed graph.*

Description

Determines if a half-trek system exists in the mixed graph.

Usage

```
getHalfTrekSystem(this, fromNodes, toNodes, ...)

## S3 method for class 'MixedGraph'
getHalfTrekSystem(
  this,
  fromNodes,
  toNodes,
  avoidLeftNodes = integer(0),
  avoidRightNodes = integer(0),
  avoidRightEdges = integer(0),
  ...
)
```

```
## S3 method for class 'MixedGraph'
getTrekSystem(
  this,
  fromNodes,
  toNodes,
  avoidLeftNodes = integer(0),
  avoidRightNodes = integer(0),
  avoidLeftEdges = integer(0),
  avoidRightEdges = integer(0),
  ...
)
```

Arguments

`this`	the mixed graph object
`fromNodes`	the nodes from which the half-trek system should start. If length(fromNodes) > length(toNodes) will find if there exists any half-trek system from any subset of fromNodes of size length(toNodes) to toNodes.
`toNodes`	the nodes where the half-trek system should end.
`...`	ignored.
`avoidLeftNodes`	a collection of nodes to avoid on the left
`avoidRightNodes`	a collection of nodes to avoid on the right
`avoidRightEdges`	a collection of edges between observed noes in the graph that should not be used on any right hand side of any trek in the trek system.
`avoidLeftEdges`	a collection of edges between observed nodes in the graph that should not be used on any right hand side of any trek in the trek system.

Value

a list with two named components, `systemExists` (TRUE if a system exists, FALSE otherwise) and `activeFrom` (the subset of fromNodes from which the maximal half-trek system was started).

`getMaxFlow` *Size of largest HT system Y satisfying the HTC for a node v except perhaps having |getParents(v)| < |Y|.*

Description

For an input mixed graph H, constructs the Gflow graph as described in Foygel et al. (2012) for a subgraph G of H. A max flow algorithm is then run on Gflow to determine the largest half-trek system in G to a particular node's getParents given a set of allowed nodes. Here G should consist of a bidirected part and nodes which are not in the bidirected part but are a parent of some node in the bidirected part. G should contain the node for which to compute the max flow.

Usage

```
getMaxFlow(L, O, allowedNodes, biNodes, inNodes, node)
```

Arguments

`L`	Adjacency matrix for the directed part of the path diagram/mixed graph; an edge pointing from i to j is encoded as L[i,j]=1 and the lack of an edge between i and j is encoded as L[i,j]=0. There should be no directed self loops, i.e. no i such that L[i,i]=1.
`O`	Adjacency matrix for the bidirected part of the path diagram/mixed graph. Edges are encoded as for the L parameter. Again there should be no self loops. Also this matrix will be coerced to be symmetric so it is only necessary to specify an edge once, i.e. if O[i,j]=1 you may, but are not required to, also have O[j,i]=1.
`allowedNodes`	the set of allowed nodes.
`biNodes`	the set of nodes in the subgraph G which are part of the bidirected part.
`inNodes`	the nodes of the subgraph G which are not in the bidirected part but are a parent of some node in the bidirected component.
`node`	the node (as an integer) for which the maxflow the largest half trek system

Value

See title.

References

Foygel, R., Draisma, J., and Drton, M. (2012) Half-trek criterion for generic identifiability of linear structural equation models. *Ann. Statist.* 40(3): 1682-1713.

`getMixedCompForNode` *Get the mixed component of a node in a mixed subgraph.*

Description

For an input mixed graph H and set of nodes A, let GA be the subgraph of H on the nodes A. This function returns the mixed component of GA containing a specified node.

Usage

```
getMixedCompForNode(dG, bG, subNodes, node)
```

Arguments

`dG`	a directed graph representing the directed part of the mixed graph.
`bG`	an undirected graph representing the undirected part of the mixed graph.
`subNodes`	an ancestral set of nodes in the mixed graph, this set should include the node for which the mixed component sould be found.
`node`	the node for which the mixed component is found.

Value

a list with two named elements: biNodes - the nodes of the mixed graph in the biDirected component containing nodeName w.r.t the ancestral set of nodes inNodes - the nodes in the graph which are not part of biNodes but which are a parent of some node in biNodes.

---

getMixedGraph — *Get the corresponding mixed graph*

Description

Only works for graphs where the latent nodes are source nodes

Usage

```
getMixedGraph(this, ...)

## S3 method for class 'LatentDigraph'
getMixedGraph(this, ...)
```

Arguments

`this`	the LatentDigraph object
`...`	ignored

---

getParents — *Get getParents of nodes in a graph.*

Description

Get the getParents of a collection of nodes in a graph g, the getParents DO include the input nodes themselves.

Usage

```
getParents(g, nodes)
```

Arguments

`g`	the graph (as an igraph).
`nodes`	the nodes in the graph of which to get the getParents.

Value

a sorted vector of all parent nodes.

getSiblings *Get getSiblings of nodes in a graph.*

Description

Get the getSiblings of a collection of nodes in a graph g, the getSiblings DO include the input nodes themselves.

Usage

```
getSiblings(g, nodes)
```

Arguments

g	the graph (as an igraph).
nodes	the nodes in the graph of which to get the getSiblings.

Value

a sorted vector of all getSiblings of nodes.

getTrekSystem *Determines if a trek system exists in the mixed graph.*

Description

Determines if a trek system exists in the mixed graph.

Usage

```
getTrekSystem(
  this,
  fromNodes,
  toNodes,
  avoidLeftNodes = integer(0),
  avoidRightNodes = integer(0),
  avoidLeftEdges = integer(0),
  avoidRightEdges = integer(0),
  ...
)

## S3 method for class 'LatentDigraphFixedOrder'
getTrekSystem(
  this,
  fromNodes,
```

```
    toNodes,
    avoidLeftNodes = integer(0),
    avoidRightNodes = integer(0),
    avoidLeftEdges = integer(0),
    avoidRightEdges = integer(0),
    ...
  )

  ## S3 method for class 'LatentDigraph'
  getTrekSystem(
    this,
    fromNodes,
    toNodes,
    avoidLeftNodes = integer(0),
    avoidRightNodes = integer(0),
    avoidLeftEdges = integer(0),
    avoidRightEdges = integer(0),
    ...
  )
```

Arguments

`this`	the graph object
`fromNodes`	the start nodes
`toNodes`	the end nodes
`avoidLeftNodes`	a collection of nodes to avoid on the left
`avoidRightNodes`	a collection of nodes to avoid on the right
`avoidLeftEdges`	a collection of edges between observed nodes in the graph that should not be used on any right hand side of any trek in the trek system.
`avoidRightEdges`	a collection of edges between observed noes in the graph that should not be used on any right hand side of any trek in the trek system.
`...`	ignored

`globalID` *Determines whether a mixed graph is globally identifiable.*

Description

Uses the criterion in Theorem 2 of the paper by Drton, Foygel and Sullivant (2011) to determine whether a mixed graph is globally identifiable.

Usage

```
globalID(graph)
```

Arguments

graph	a MixedGraph object representing the mixed graph.

Value

TRUE if the graph is globally identifiable, FALSE otherwise.

References

Drton, M., Barber, R. F., and Sullivant S. (2011). Half-Trek Criterion for Identifiability of Latent Variable Models. Ann. Statist. 39 (2011), no. 2, 865–886 <doi:10.1214/10-AOS859>.

---

graphID *Identifiability of linear structural equation models.*

Description

NOTE: graphID has been deprecated, use semID instead.

This function checks global and generic identifiability of linear structural equation models. For generic identifiability the function checks a sufficient criterion as well as a necessary criterion but this check may be inconclusive.

Usage

```
graphID(
  L,
  O,
  output.type = "matrix",
  file.name = NULL,
  decomp.if.acyclic = TRUE,
  test.globalID = TRUE,
  test.genericID = TRUE,
  test.nonID = TRUE
)
```

Arguments

L	Adjacency matrix for the directed part of the path diagram/mixed graph; an edge pointing from i to j is encoded as L[i,j]=1 and the lack of an edge between i and j is encoded as L[i,j]=0. There should be no directed self loops, i.e. no i such that L[i,i]=1.
O	Adjacency matrix for the bidirected part of the path diagram/mixed graph. Edges are encoded as for the L parameter. Again there should be no self loops. Also this matrix will be coerced to be symmetric so it is only necessary to specify an edge once, i.e. if O[i,j]=1 you may, but are not required to, also have O[j,i]=1.

`output.type` A character string indicating whether output is printed ('matrix'), saved to a file ('file'), or returned as a list ('list') for further processing in R.

`file.name` A character string naming the output file.

`decomp.if.acyclic` A logical value indicating whether an input graph that is acyclic is to be decomposed before applying identifiability criteria.

`test.globalID` A logical value indicating whether or not global identifiability is checked.

`test.genericID` A logical value indicating whether or not a sufficient condition for generic identifiability is checked.

`test.nonID` A logical value indicating whether or not a condition implying generic non-identifiability is checked.

Value

A list or printed matrix indicating the identifiability status of the linear SEM given by the input graph. Optionally the graph's components are listed.

With output.type = 'list', the function returns a list of components for the graph. Each list entry is again a list that indicates first which nodes form the component and second whether the component forms a mixed graph that is acyclic. The next entries in the list show HTC-identifiable nodes, meaning nodes v for which the coefficients for all the directed edges pointing to v can be identified using the methods from Foygel et al. (2012). The HTC-identifiable nodes are listed in the order in which they are found by the recursive identification algorithm. The last three list entries are logical values that indicate whether or not the graph component is generically identifiable, globally identifiable or not identifiable; compare Drton et al. (2011) and Foygel et al. (2012). In the latter case the Jacobian of the parametrization does not have full rank.

With output.type = 'matrix', a summary of the above information is printed.

References

Drton, M., Foygel, R., and Sullivant, S. (2011) Global identifiability of linear structural equation models. *Ann. Statist.* 39(2): 865-886.

Foygel, R., Draisma, J., and Drton, M. (2012) Half-trek criterion for generic identifiability of linear structural equation models. *Ann. Statist.* 40(3): 1682-1713.

Examples

```
## Not run:
L = t(matrix(
  c(0, 1, 0, 0, 0,
    0, 0, 1, 0, 0,
    0, 0, 0, 1, 0,
    0, 0, 0, 0, 1,
    0, 0, 0, 0, 0), 5, 5))
O = t(matrix(
  c(0, 0, 1, 1, 0,
    0, 0, 0, 1, 1,
    0, 0, 0, 0, 0,
    0, 0, 0, 0, 0,
```

```
    0, 0, 0, 0, 0), 5, 5))
O=O+t(O)
graphID(L,O)


## Examples from Foygel, Draisma & Drton (2012)
demo(SEMID)

## End(Not run)
```

`graphID.ancestralID` *Determine generic identifiability of an acyclic mixed graph using ancestral decomposition.*

Description

For an input, acyclic, mixed graph attempts to determine if the graph is generically identifiable using decomposition by ancestral subsets. See algorithm 1 of Drton and Weihs (2015).

Usage

```
graphID.ancestralID(L, O)
```

Arguments

`L`	Adjacency matrix for the directed part of the path diagram/mixed graph; an edge pointing from i to j is encoded as L[i,j]=1 and the lack of an edge between i and j is encoded as L[i,j]=0. There should be no directed self loops, i.e. no i such that L[i,i]=1.
`O`	Adjacency matrix for the bidirected part of the path diagram/mixed graph. Edges are encoded as for the L parameter. Again there should be no self loops. Also this matrix will be coerced to be symmetric so it is only necessary to specify an edge once, i.e. if O[i,j]=1 you may, but are not required to, also have O[j,i]=1.

Value

The vector of nodes that could be determined to be generically identifiable using the above algorithm.

References

Drton, M. and Weihs, L. (2015) Generic Identifiability of Linear Structural Equation Models by Ancestor Decomposition. arXiv 1504.02992

graphID.decompose *Determine generic identifiability by Tian Decomposition and HTC*

Description

Split a graph into mixed Tian components and solve each separately using the HTC.

Usage

```
graphID.decompose(
  L,
  O,
  decomp.if.acyclic = TRUE,
  test.globalID = TRUE,
  test.genericID = TRUE,
  test.nonID = TRUE
)
```

Arguments

`L` Adjacency matrix for the directed part of the path diagram/mixed graph; an edge pointing from i to j is encoded as L[i,j]=1 and the lack of an edge between i and j is encoded as L[i,j]=0. There should be no directed self loops, i.e. no i such that L[i,i]=1.

`O` Adjacency matrix for the bidirected part of the path diagram/mixed graph. Edges are encoded as for the L parameter. Again there should be no self loops. Also this matrix will be coerced to be symmetric so it is only necessary to specify an edge once, i.e. if O[i,j]=1 you may, but are not required to, also have O[j,i]=1.

`decomp.if.acyclic` A logical value indicating whether an input graph that is acyclic is to be decomposed before applying identifiability criteria.

`test.globalID` A logical value indicating whether or not global identifiability is checked.

`test.genericID` A logical value indicating whether or not a sufficient condition for generic identifiability is checked.

`test.nonID` A logical value indicating whether or not a condition implying generic non-identifiability is checked.

Value

A list with two named components:

1. Components - a list of lists. Each list represents one mixed Tian component of the graph. Each list contains named components corresponding to which nodes are in the component and results of various tests of identifiability on the component (see the parameter descriptions).

2. Decomp - true if a decomposition occured, false if not.

`graphID.genericID` *Determine generic identifiability of a mixed graph.*

Description

If the directed part of input graph is cyclic then will check for generic identifiability using the half-trek criterion. Otherwise will use the a slightly stronger version of the half-trek criterion using ancestor decompositions.

Usage

```
graphID.genericID(L, O)
```

Arguments

`L`	Adjacency matrix for the directed part of the path diagram/mixed graph; an edge pointing from i to j is encoded as L[i,j]=1 and the lack of an edge between i and j is encoded as L[i,j]=0. There should be no directed self loops, i.e. no i such that L[i,i]=1.
`O`	Adjacency matrix for the bidirected part of the path diagram/mixed graph. Edges are encoded as for the L parameter. Again there should be no self loops. Also this matrix will be coerced to be symmetric so it is only necessary to specify an edge once, i.e. if O[i,j]=1 you may, but are not required to, also have O[j,i]=1.

Value

The vector of nodes that could be determined to be generically identifiable.

References

Foygel, R., Draisma, J., and Drton, M. (2012) Half-trek criterion for generic identifiability of linear structural equation models. *Ann. Statist.* 40(3): 1682-1713.

Drton, M. and Weihs, L. (2015) Generic Identifiability of Linear Structural Equation Models by Ancestor Decomposition. arXiv 1504.02992

`graphID.htcID` *Determines if a mixed graph is HTC-identifiable.*

Description

Uses the half-trek criterion of Foygel, Draisma, and Drton (2013) to check if an input mixed graph is generically identifiable.

Usage

```
graphID.htcID(L, O)
```

Arguments

`L` Adjacency matrix for the directed part of the path diagram/mixed graph; an edge pointing from i to j is encoded as L[i,j]=1 and the lack of an edge between i and j is encoded as L[i,j]=0. There should be no directed self loops, i.e. no i such that L[i,i]=1.

`O` Adjacency matrix for the bidirected part of the path diagram/mixed graph. Edges are encoded as for the L parameter. Again there should be no self loops. Also this matrix will be coerced to be symmetric so it is only necessary to specify an edge once, i.e. if O[i,j]=1 you may, but are not required to, also have O[j,i]=1.

Value

The vector of HTC-identifiable nodes.

References

Foygel, R., Draisma, J., and Drton, M. (2012) Half-trek criterion for generic identifiability of linear structural equation models. *Ann. Statist.* 40(3): 1682-1713.

`graphID.main` *Helper function to handle a graph component.*

Description

Calls the other functions that determine identifiability status.

Usage

```
graphID.main(
  L,
  O,
  test.globalID = TRUE,
  test.genericID = TRUE,
  test.nonID = TRUE
)
```

Arguments

`L` Adjacency matrix for the directed part of the path diagram/mixed graph; an edge pointing from i to j is encoded as L[i,j]=1 and the lack of an edge between i and j is encoded as L[i,j]=0. There should be no directed self loops, i.e. no i such that L[i,i]=1.

`O` Adjacency matrix for the bidirected part of the path diagram/mixed graph. Edges are encoded as for the L parameter. Again there should be no self loops. Also this matrix will be coerced to be symmetric so it is only necessary to specify an edge once, i.e. if O[i,j]=1 you may, but are not required to, also have O[j,i]=1.

`test.globalID`	A logical value indicating whether or not global identifiability is checked.
`test.genericID`	A logical value indicating whether or not a sufficient condition for generic identifiability is checked.
`test.nonID`	A logical value indicating whether or not a condition implying generic non-identifiability is checked.

Value

A list containing named components of the results of various tests desired based on the input parameters.

`graphID.nonHtcID` *Check for generic infinite-to-one via the half-trek criterion.*

Description

Checks if a mixed graph is infinite-to-one using the half-trek criterion presented by Foygel, Draisma, and Drton (2012).

Usage

```
graphID.nonHtcID(L, O)
```

Arguments

`L`	Adjacency matrix for the directed part of the path diagram/mixed graph; an edge pointing from i to j is encoded as L[i,j]=1 and the lack of an edge between i and j is encoded as L[i,j]=0. There should be no directed self loops, i.e. no i such that L[i,i]=1.
`O`	Adjacency matrix for the bidirected part of the path diagram/mixed graph. Edges are encoded as for the L parameter. Again there should be no self loops. Also this matrix will be coerced to be symmetric so it is only necessary to specify an edge once, i.e. if O[i,j]=1 you may, but are not required to, also have O[j,i]=1.

Value

TRUE if the graph could be determined to be generically non-identifiable, FALSE if this test was inconclusive.

References

Foygel, R., Draisma, J., and Drton, M. (2012) Half-trek criterion for generic identifiability of linear structural equation models. *Ann. Statist.* 40(3): 1682-1713.

htcID — *Determines which edges in a mixed graph are HTC-identifiable.*

Description

Uses the half-trek criterion of Foygel, Draisma, and Drton (2012) determine which edges in a mixed graph are generically identifiable. Depending on your application it faster to use the `graphID.htcID` function instead of this one, this function has the advantage of returning additional information.

Usage

```
htcID(mixedGraph, tianDecompose = T)
```

Arguments

`mixedGraph`	a `MixedGraph` object representing the L-SEM.
`tianDecompose`	TRUE or FALSE determining whether or not the Tian decomposition should be used before running the current generic identification algorithm. In general letting this be TRUE will make the algorithm faster and more powerful.

Value

see the return value of `generalGenericID`.

References

Foygel, R., Draisma, J., and Drton, M. (2012) Half-trek criterion for generic identifiability of linear structural equation models. *Ann. Statist.* 40(3): 1682-1713.

Jin Tian. 2005. Identifying direct causal effects in linear models. In *Proceedings of the 20th national conference on Artificial intelligence - Volume 1* (AAAI'05), Anthony Cohn (Ed.), Vol. 1. AAAI Press 346-352.

htcIdentifyStep — *Perform one iteration of HTC identification.*

Description

A function that does one step through all the nodes in a mixed graph and tries to identify new edge coefficients using the existence of half-trek systems as described in Foygel, Draisma, Drton (2012).

Usage

```
htcIdentifyStep(mixedGraph, unsolvedParents, solvedParents, identifier)
```

Arguments

`mixedGraph` a MixedGraph object representing the mixed graph.

`unsolvedParents` a list whose ith index is a vector of all the parents j of i in G which for which the edge j->i is not yet known to be generically identifiable.

`solvedParents` the complement of `unsolvedParents`, a list whose ith index is a vector of all parents j of i for which the edge i->j is known to be generically identifiable (perhaps by other algorithms).

`identifier` an identification function that must produce the identifications corresponding to those in solved parents. That is `identifier` should be a function taking a single argument Sigma (any generically generated covariance matrix corresponding to the mixed graph) and returns a list with two named arguments

Lambda denote the number of nodes in `mixedGraph` as n. Then Lambda is an nxn matrix whose i,jth entry

1. equals 0 if i is not a parent of j,
2. equals NA if i is a parent of j but `identifier` cannot identify it generically,
3. equals the (generically) unique value corresponding to the weight along the edge i->j that was used to produce Sigma.

Omega just as Lambda but for the bidirected edges in the mixed graph

such that if j is in `solvedParents[[i]]` we must have that Lambda[j,i] is not NA.

Value

a list with four components:

`identifiedEdges` a matrix rx2 matrix where r is the number of edges that where identified by this function call and `identifiedEdges[i,1]` -> `identifiedEdges[i,2]` was the ith edge identified

`unsolvedParents` as the input argument but updated with any newly identified edges

`solvedParents` as the input argument but updated with any newly identified edges

`identifier` as the input argument but updated with any newly identified edges

References

Foygel, R., Draisma, J., and Drton, M. (2012) Half-trek criterion for generic identifiability of linear structural equation models. *Ann. Statist.* 40(3): 1682-1713

htr	*Get all HTR nodes from a set of nodes in a graph.*

Description

Gets all vertices in a graph that are half-trek reachable from a set of nodes. WARNING: Often the half-trek reachable nodes from a vertex v are defined to not include the vertex v or its getSiblings. We DO NOT follow this convention, the returned set will include input nodes and their getSiblings.

Usage

```
htr(dG, bG, nodes)
```

Arguments

`dG`	a directed graph representing the directed part of the mixed graph.
`bG`	an undirected graph representing the undirected part of the mixed graph.
`nodes`	the nodes in the graph of which to get the HTR nodes.

Value

a sorted list of all half-trek reachable nodes.

htrFrom	*Half trek reachable nodes.*

Description

Half trek reachable nodes.

Usage

```
htrFrom(this, nodes, ...)

## S3 method for class 'MixedGraph'
htrFrom(
  this,
  nodes,
  avoidLeftNodes = integer(0),
  avoidRightNodes = integer(0),
  ...
)
```

Arguments

`this`	the mixed graph object
`nodes`	the nodes from which to get all half-trek reachable nodes.
`...`	ignored.
`avoidLeftNodes`	a collection of nodes to avoid on the left
`avoidRightNodes`	a collection of nodes to avoid on the right

Value

a vector of all nodes half-trek reachable from node.

`inducedSubgraph` *Get the induced subgraph on a collection of nodes*

Description

Get the induced subgraph on a collection of nodes

Usage

```
inducedSubgraph(this, nodes, ...)

## S3 method for class 'LatentDigraph'
inducedSubgraph(this, nodes, ...)

## S3 method for class 'MixedGraph'
inducedSubgraph(this, nodes, ...)
```

Arguments

`this`	the graph object
`nodes`	the nodes on which to create the induced subgraph.
`...`	ignored.

isSibling *Are two nodes siblings?*

Description

Are two nodes siblings?

Usage

```
isSibling(this, node1, node2, ...)

## S3 method for class 'MixedGraph'
isSibling(this, node1, node2, ...)
```

Arguments

this	the mixed graph object
node1	a node
node2	a second node
...	ignored.

Value

TRUE if the nodes are siblings in the graph, FALSE otherwise

L *Get directed adjacency matrix.*

Description

Get directed adjacency matrix.

Usage

```
L(this, ...)

## S3 method for class 'LatentDigraphFixedOrder'
L(this, ...)

## S3 method for class 'LatentDigraph'
L(this, ...)

## S3 method for class 'MixedGraph'
L(this, ...)
```

Arguments

`this`	the graph object
`...`	ignored.

`LatentDigraph` *Construct a LatentDigraph object*

Description

Creates an object representing a latent factor graph. The methods that are currently available to be used on the latent factor graph include

1. numObserved
2. numLatents
3. numNodes
4. toIn
5. toEx
6. L
7. observedNodes
8. latentNodes
9. parents
10. children
11. ancestors
12. descendants
13. trFrom
14. getTrekSystem
15. inducedSubgraph
16. stronglyConnectedComponent
17. plot
18. observedParents
19. getMixedGraph

see the individual function documentation for more information.

Usage

```
LatentDigraph(L = matrix(0,1,1),
                     observedNodes = seq(1, length = nrow(L)),
                     latentNodes = integer(0))
```

Arguments

`L`	see `graphID` for the appropriate form of L.
`observedNodes`	a vector of positive integers representing the vertex numbers of the observed nodes. These will correspond, in order, to the first length(observedNodes) rows of L.
`latentNodes`	a vector of positive integers representing the vertex numbers of the latent nodes. These will correspond, in order, to the last length(latentNodes) rows of L.

Value

An object representing the LatentDigraph

`LatentDigraphFixedOrder`

Construct LatentDigraphFixedOrder object

Description

Creates an object representing a directed graph with some number of nodes which are latent (unobserved).

Usage

```
LatentDigraphFixedOrder(L = matrix(0,1,1), numObserved = nrow(L))
```

Arguments

`L`	see `graphID` for the appropriate form of L. The first numObserved rows of L correspond to the observed nodes in the graph, all other nodes are considered unobserved.
`numObserved`	a non-negative integer representing the number of observed nodes in the graph.

Value

An object representing the LatentDigraphFixedOrder `latentDigraphHasSimpleNumbering` *Checks that a LatentDigraph has appropriate node numbering*

Description

Checks that the input latent digraph has nodes numbered from 1 to latentDigraph$numObserved()+latentDigraph$numLatents
The first latentDigraph$numObserved() nodes correspond to the observed nodes in the graph, all other nodes are considered unobserved. Throws an error if this is not true.

Usage

```
latentDigraphHasSimpleNumbering(graph)
```

Arguments

graph	a `LatentDigraph` object representing the latent-factor graph. All latent nodes in this graph should be source nodes (i.e. have no parents).

`latentNodes` *Get all latent nodes in the graph.*

Description

Get all latent nodes in the graph.

Usage

```
latentNodes(this, ...)

## S3 method for class 'LatentDigraph'
latentNodes(this, ...)
```

Arguments

this	the graph object
...	ignored

`lfhtcID` *Determines which edges in a latent digraph are LF-HTC-identifiable.*

Description

Uses the latent-factor half-trek criterion to determine which edges in a latent digraph are generically identifiable.

Usage

```
lfhtcID(graph)
```

Arguments

`graph` a `LatentDigraph` object representing the latent-factor graph. All latent nodes in this graph should be source nodes (i.e. have no parents).

Value

returns a list with 8 components:

`solvedParents` a list whose ith element contains a vector containing the subsets of parents of node i for which the edge j->i could be shown to be generically identifiable.

`unsolvedParents` as for `solvedParents` but for the unsolved parents.

`identifier` a function that takes a (generic) covariance matrix corresponding to the graph and identifies the edges parameters from solvedParents and solvedSiblings. See `htcIdentifyStep` for a more in-depth discussion of identifier functions.

`graph` a latent digraph object of the graph.

`call` the call made to this function.

`activeFroms` list. If node i is solved then the ith index is a vector containing the nodes Y otherwise it is empty.

`Zs` list. If node i is solved then the ith index is a vector containing the nodes Z otherwise it is empty.

`Ls` list. If node i is solved then the ith index is a vector containing the nodes L otherwise it is empty.

References

Barber, R. F., Drton, M., Sturma, N., and Weihs L. (2022). Half-Trek Criterion for Identifiability of Latent Variable Models. Ann. Statist. 50(6):3174–3196. <doi:10.1214/22-AOS2221>.

lfhtcIdentifyStep — *Perform one iteration of latent-factor HTC identification.*

Description

A function that does one step through all the nodes in a latent-factor graph and tries to identify new edge coefficients using the existence of latent-factor half-trek systems.

Usage

```
lfhtcIdentifyStep(
  graph,
  unsolvedParents,
  solvedParents,
  activeFroms,
  Zs,
  Ls,
  identifier,
  subsetSizeControl = Inf
)
```

Arguments

`graph`	a `LatentDigraph` object representing the latent-factor graph. All latent nodes in this graph should be source nodes (i.e. have no parents).
`unsolvedParents`	a list whose ith index is a vector of all the parents j of i in the graph which for which the edge j->i is not yet known to be generically identifiable.
`solvedParents`	the complement of `unsolvedParents`, a list whose ith index is a vector of all parents j of i for which the edge i->j is known to be generically identifiable (perhaps by other algorithms).
`activeFroms`	list. If node i is solved then the ith index is a vector containing the nodes Y otherwise it is empty.
`Zs`	list. If node i is solved then the ith index is a vector containing the nodes Z otherwise it is empty.
`Ls`	list. If node i is solved then the ith index is a vector containing the nodes Z otherwise it is empty.
`identifier`	an identification function that must produce the identifications corresponding to those in solved parents. That is `identifier` should be a function taking a single argument Sigma (any generically generated covariance matrix corresponding to the latent-factor graph) and returns a list with two named arguments
`subsetSizeControl`	the largest subset of latent nodes to consider.

Value

a list with four components:

`identifiedEdges` a matrix rx2 matrix where r is the number of edges that where identified by this function call and `identifiedEdges[i,1]` -> `identifiedEdges[i,2]` was the ith edge identified

`unsolvedParents` as the input argument but updated with any newly identified edges

`solvedParents` as the input argument but updated with any newly identified edges

`identifier` as the input argument but updated with any newly identified edges

`activeFroms` as the input argument but updated with any newly solved node

`Zs` as the input argument but updated with any newly solved node

`Ls` as the input argument but updated with any newly solved node

References

Barber, R. F., Drton, M., Sturma, N., and Weihs L. (2022). Half-Trek Criterion for Identifiability of Latent Variable Models. *arXiv preprint* arXiv:2201.04457

`mID` *Check M-Identifiability.*

Description

Check M-Identifiability.

Usage

```
mID(lambda, maxCard = length(observedNodes))
```

Arguments

`lambda`	adjacency matrix with number of cols = number of latent nodes, number of rows = number of observed nodes
`maxCard`	(optional) maximum size of set W

Value

a list consisting of a Boolean, whether the graph is sign-identifiable and if yes, a list consisting of the sets

References

Sturma, N., Kranzlmüller, M., Portakal, I., and Drton, M. (2025) Matching Criterion for Identifiability in Sparse Factor Analysis. arXiv:2502.02986

`MixedGraph` *Construct MixedGraph object*

Description

Creates an object representing a mixed graph. The methods that are currently available to be used on the mixed graph include

1. ancestors
2. descendants
3. parents
4. siblings
5. isSibling
6. htrFrom
7. trFrom
8. getHalfTrekSystem
9. getTrekSystem
10. inducedSubgraph
11. L
12. O
13. nodes
14. numNodes
15. stronglyConnectedComponent
16. tianComponent
17. tianDecompose

see the individual function documentation for more information.

Usage

```
MixedGraph(
  L = matrix(0, 1, 1),
  O = matrix(0, 1, 1),
  vertexNums = seq(1, length = nrow(L))
)
```

Arguments

L	see graphID for the appropriate form of L.
O	as for L.
vertexNums	the labeling of the vertices in the graph in the order of the rows of L and O. Labels must be positive integers.

Value

An object representing the MixedGraph

`mixedGraphHasSimpleNumbering` *Checks that a MixedGraph has appropriate node numbering*

Description

Checks that the input mixed graph has vertices are numbered from 1 to mixedGraph$numNodes(). Throws an error if they are not.

Usage

```
mixedGraphHasSimpleNumbering(mixedGraph)
```

Arguments

`mixedGraph`	the mixed graph object

`nodes` *Get all nodes in the graph.*

Description

Get all nodes in the graph.

Usage

```
nodes(this, ...)

## S3 method for class 'MixedGraph'
nodes(this, ...)
```

Arguments

`this`	the mixed graph object
`...`	ignored.

numLatents — *Number of latent nodes in the graph.*

Description

Number of latent nodes in the graph.

Usage

```
numLatents(this, ...)

## S3 method for class 'LatentDigraphFixedOrder'
numLatents(this, ...)

## S3 method for class 'LatentDigraph'
numLatents(this, ...)
```

Arguments

`this`	the graph object
`...`	ignored

numNodes — *Number of nodes in the graph.*

Description

Number of nodes in the graph.

Usage

```
numNodes(this, ...)

## S3 method for class 'LatentDigraphFixedOrder'
numNodes(this, ...)

## S3 method for class 'LatentDigraph'
numNodes(this, ...)

## S3 method for class 'MixedGraph'
numNodes(this, ...)
```

Arguments

`this`	the graph object
`...`	ignored.

numObserved *Number of observed nodes in the graph.*

Description

Number of observed nodes in the graph.

Usage

```
numObserved(this, ...)

## S3 method for class 'LatentDigraphFixedOrder'
numObserved(this, ...)

## S3 method for class 'LatentDigraph'
numObserved(this, ...)
```

Arguments

this	the graph object
...	ignored

O *Get adjacency matrix for bidirected part.*

Description

Get adjacency matrix for bidirected part.

Usage

```
O(this, ...)

## S3 method for class 'MixedGraph'
O(this, ...)
```

Arguments

this	the mixed graph object
...	ignored.

observedNodes	*Get all observed nodes in the graph.*

Description

Get all observed nodes in the graph.

Usage

```
observedNodes(this, ...)

## S3 method for class 'LatentDigraph'
observedNodes(this, ...)
```

Arguments

`this`	the graph object
`...`	ignored

observedParents	*Get the observed parents on a collection of nodes*

Description

Get the observed parents on a collection of nodes

Usage

```
observedParents(this, nodes, ...)

## S3 method for class 'LatentDigraph'
observedParents(this, nodes, ...)
```

Arguments

`this`	the graph object
`nodes`	the nodes on which to get the observed parents
`...`	ignored

`parents` *All parents of a collection of nodes.*

Description

Returns all parents of the collection (does not necessarily include the input nodes themselves unless they are parents of one another).

Usage

```
parents(this, nodes, ...)

## S3 method for class 'LatentDigraphFixedOrder'
parents(this, nodes, includeObserved = T, includeLatents = T, ...)

## S3 method for class 'LatentDigraph'
parents(this, nodes, includeObserved = T, includeLatents = T, ...)

## S3 method for class 'MixedGraph'
parents(this, nodes, ...)
```

Arguments

`this`	the graph object.
`nodes`	nodes the nodes of which to find the parents.
`...`	ignored.
`includeObserved`	if TRUE includes observed nodes in the returned set.
`includeLatents`	if TRUE includes latent nodes in the returned set.

Value

the observed parents.

`plot.LatentDigraph` *Plots the latent digraph*

Description

Plots the latent digraph

Plots the mixed graph

Usage

```
## S3 method for class 'LatentDigraph'
plot(x, ...)

## S3 method for class 'MixedGraph'
plot(x, ...)
```

Arguments

`x`	the mixed graph object
`...`	additional plotting arguments. Currently ignored.

`plotLatentDigraph` *Plot a latent factor graph*

Description

Given an adjacency matrix representing the directed edges in a latent factor graph, plots a representation of the graph. The latent nodes should come last in L and the vertex labels should only be given for the observed nodes.

Usage

```
plotLatentDigraph(L, observedNodes, latentNodes, main = "")
```

Arguments

`L`	Adjacency matrix for the directed part of the path diagram/mixed graph; an edge pointing from i to j is encoded as L[i,j]=1 and the lack of an edge between i and j is encoded as L[i,j]=0. There should be no directed self loops, i.e. no i such that L[i,i]=1.
`observedNodes`	a vector of positive integers representing the vertex numbers of the observed nodes. These will correspond, in order, to the first length(observedNodes) rows of L.
`latentNodes`	a vector of positive integers representing the vertex numbers of the latent nodes. These will correspond, in order, to the last length(latentNodes) rows of L.
`main`	the plot title.

Value

An object representing the LatentDigraph

plotMixedGraph *Plot a mixed graph*

Description

Given adjacency matrices representing the directed and bidirected portions of a mixed graph, plots a representation of the graph.

Usage

```
plotMixedGraph(L, O, main = "", vertexLabels = 1:nrow(L))
```

Arguments

L	Adjacency matrix for the directed part of the path diagram/mixed graph; an edge pointing from i to j is encoded as L[i,j]=1 and the lack of an edge between i and j is encoded as L[i,j]=0. There should be no directed self loops, i.e. no i such that L[i,i]=1.
O	Adjacency matrix for the bidirected part of the path diagram/mixed graph. Edges are encoded as for the L parameter. Again there should be no self loops. Also this matrix will be coerced to be symmetric so it is only necessary to specify an edge once, i.e. if O[i,j]=1 you may, but are not required to, also have O[j,i]=1.
main	the plot title.
vertexLabels	labels to use for the vertices.

print.extmIDresult *Prints a extmIDresult object*

Description

Prints a extmIDresult object as returned by `extmID`. Invisibly returns its argument via `invisible(x)` as most print functions do.

Usage

```
## S3 method for class 'extmIDresult'
print(x, ...)
```

Arguments

x	the extmIDresult object
...	optional parameters, currently unused.

print.GenericIDResult *Prints a GenericIDResult object*

Description

Prints a GenericIDResult object as returned by `generalGenericID`. Invisibly returns its argument via `invisible(x)` as most print functions do.

Usage

```
## S3 method for class 'GenericIDResult'
print(x, ...)
```

Arguments

`x`	the GenericIDResult object
`...`	optional parameters, currently unused.

print.LfhtcIDResult *Prints a LfhtcIDResult object*

Description

Prints a LfhtcIDResult object as returned by `lfhtcID`. Invisibly returns its argument via `invisible(x)` as most print functions do.

Usage

```
## S3 method for class 'LfhtcIDResult'
print(x, ...)
```

Arguments

`x`	the LfhtcIDResult object
`...`	optional parameters, currently unused.

`print.mIDresult` *Prints a mIDresult object*

Description

Prints a mIDresult object as returned by `mID`. Invisibly returns its argument via `invisible(x)` as most print functions do.

Usage

```
## S3 method for class 'mIDresult'
print(x, ...)
```

Arguments

`x`	the mIDresult object
`...`	optional parameters, currently unused.

`print.SEMIDResult` *Prints a SEMIDResult object*

Description

Prints a SEMIDResult object as returned by `semID`. Invisibly returns its argument via `invisible(x)` as most print functions do.

Usage

```
## S3 method for class 'SEMIDResult'
print(x, ...)
```

Arguments

`x`	the SEMIDResult object
`...`	optional parameters, currently unused.

print.ZUTAresult *Prints a ZUTAresult object*

Description

Prints a ZUTAresult object as returned by ZUTA. Invisibly returns its argument via invisible(x) as most print functions do.

Usage

```
## S3 method for class 'ZUTAresult'
print(x, ...)
```

Arguments

x	the ZUTAresult object
...	optional parameters, currently unused.

semID *Identifiability of linear structural equation models.*

Description

This function can be used to check global and generic identifiability of linear structural equation models (L-SEMs). In particular, this function takes a MixedGraph object corresponding to the L-SEM and checks different conditions known for global and generic identifiability.

Usage

```
semID(
  mixedGraph,
  testGlobalID = TRUE,
  testGenericNonID = TRUE,
  genericIdStepFunctions = list(htcIdentifyStep),
  tianDecompose = TRUE
)
```

Arguments

mixedGraph	a MixedGraph object representing the L-SEM.
testGlobalID	TRUE or FALSE if the graph should be tested for global identifiability. This uses the globalID function.

`testGenericNonID`
: TRUE of FALSE if the graph should be tested for generic non-identifiability, that is, if for every generic choice of parameters for the L-SEM there are infinitely many other choices that lead to the same covariance matrix. This currently uses the `graphID.nonHtcID` function.

`genericIdStepFunctions`
: a list of the generic identifier step functions that should be used for testing generic identifiability. See `generalGenericID` for a discussion of such functions. If this list is empty then generic identifiability is not tested. By default this will (only) run the half-trek criterion (see `htcIdentifyStep`) for generic identifiability.

`tianDecompose`
: TRUE or FALSE if the mixed graph should be Tian decomposed before running the identification algorithms (when appropriate). In general letting this be TRUE will make the algorithm faster and more powerful. Note that this is a version of the Tian decomposition that works also with cyclic graphs.

Value

returns an object of class 'SEMIDResult,' this object is just a list with 6 components:

`isGlobalID` If testGlobalID == TRUE, then TRUE or FALSE if the graph is globally identifiable. If testGlobalID == FALSE then NA.

`isGenericNonID` If testGenericNonID == TRUE, then TRUE if the graph is generically non-identifiable or FALSE the test is inconclusive. If testGenericNonID == FALSE then NA.

`genericIDResult` If length(genericIdStepFunctions) != 0 then a `GenericIDResult` object as returned by `generalGenericID`. Otherwise a list of length 0.

`mixedGraph` the inputted mixed graph object.

`tianDecompose` the argument tianDecompose.

`call` the call made to this function.

Examples

```
## Not run:
L = t(matrix(
  c(0, 1, 0, 0, 0,
    0, 0, 1, 0, 0,
    0, 0, 0, 1, 0,
    0, 0, 0, 0, 1,
    0, 0, 0, 0, 0), 5, 5))
O = t(matrix(
  c(0, 0, 1, 1, 0,
    0, 0, 0, 1, 1,
    0, 0, 0, 0, 0,
    0, 0, 0, 0, 0,
    0, 0, 0, 0, 0), 5, 5))
O = O + t(O)
graph = MixedGraph(L,O)
semID(graph)
```

```
## Examples from Foygel, Draisma & Drton (2012)
demo(SEMID)

## End(Not run)
```

`siblings` *All siblings of a collection of nodes*

Description

All siblings of a collection of nodes

Usage

```
siblings(this, nodes, ...)

## S3 method for class 'MixedGraph'
siblings(this, nodes, ...)
```

Arguments

`this`	the mixed graph object
`nodes`	a vector of nodes of which to find the siblings.
`...`	ignored.

Value

a vector of all of the siblings.

`stronglyConnectedComponent` *Strongly connected component*

Description

Get the strongly connected component for a node i in the graph the graph.

Usage

```
stronglyConnectedComponent(this, node, ...)

## S3 method for class 'LatentDigraphFixedOrder'
stronglyConnectedComponent(this, node, ...)

## S3 method for class 'LatentDigraph'
stronglyConnectedComponent(this, node, ...)

## S3 method for class 'MixedGraph'
stronglyConnectedComponent(this, node, ...)
```

Arguments

this	the graph object
node	the node for which to get the strongly connected component.
...	ignored.

---

subsetsOfSize   *Returns all subsets of a certain size*

Description

For an input vector x, returns in a list, the collection of all subsets of x of size k.

Usage

```
subsetsOfSize(x, k)
```

Arguments

x	a vector from which to get subsets
k	the size of the subsets returned

Value

a list of all subsets of x of a given size k

tianComponent *Returns the Tian c-component of a node*

Description

Returns the Tian c-component of a node

Usage

```
tianComponent(this, node)

## S3 method for class 'MixedGraph'
tianComponent(this, node)
```

Arguments

`this`	the mixed graph object
`node`	the node for which to return its c-component

tianDecompose *Performs the tian decomposition on the mixed graph*

Description

Uses the Tian decomposition to break the mixed graph into c-components. These c-components are slightly different than those from Tian (2005) in that if they graph is not acyclic the bidirected components are combined whenever they are connected by a directed loop.

Usage

```
tianDecompose(this)

## S3 method for class 'MixedGraph'
tianDecompose(this)
```

Arguments

`this`	the mixed graph object

References

Jin Tian. 2005. Identifying direct causal effects in linear models. In *Proceedings of the 20th national conference on Artificial intelligence - Volume 1* (AAAI'05), Anthony Cohn (Ed.), Vol. 1. AAAI Press 346-352.

tianIdentifier *Identifies components in a tian decomposition*

Description

Creates an identification function which combines the identification functions created on a collection of c-components into a identification for the full mixed graph.

Usage

```
tianIdentifier(idFuncs, cComponents)
```

Arguments

idFuncs	a list of identifier functions for the c-components
cComponents	the c-components of the mixed graph as returned by tianDecompose.

Value

a new identifier function

tianSigmaForComponent *Globally identify the covariance matrix of a C-component*

Description

The Tian decomposition of a mixed graph G allows one to globally identify the covariance matrices Sigma' of special subgraphs of G called c-components. This function takes the covariance matrix Sigma corresponding to G and a collection of node sets which specify the c-component, and returns the Sigma' corresponding to the c-component.

Usage

```
tianSigmaForComponent(Sigma, internal, incoming, topOrder)
```

Arguments

Sigma	the covariance matrix for the mixed graph G
internal	an integer vector corresponding to the vertices of the C-component that are in the bidirected equivalence classes (if the graph is not-acyclic then these equivalence classes must be enlarged by combining two bidirected components if there are two vertices, one in each component, that are simultaneously on the same directed cycle).
incoming	the parents of vertices in internal that are not in the set internal themselves
topOrder	a topological ordering of c(internal, incoming) with respect to the graph G. For vertices in a strongly connected component the ordering is allowed to be arbitrary.

Value

the new Sigma corresponding to the c-component

---

toEx — *Transforms a vector of node indices in the internal rep. into external numbering*

Description

Transforms a vector of node indices in the internal rep. into external numbering

Usage

```
toEx(this, nodes, ...)

## S3 method for class 'LatentDigraph'
toEx(this, nodes, ...)

## S3 method for class 'MixedGraph'
toEx(this, nodes, ...)
```

Arguments

`this`	the graph object
`nodes`	the nodes to transform
`...`	ignored

---

toIn — *Transforms a vector of given node indices into their internal numbering*

Description

Transforms a vector of given node indices into their internal numbering

Usage

```
toIn(this, nodes, ...)

## S3 method for class 'LatentDigraph'
toIn(this, nodes, ...)

## S3 method for class 'MixedGraph'
toIn(this, nodes, ...)
```

Arguments

`this`	the graph object
`nodes`	the nodes to transform
`...`	ignored

`trekSeparationIdentifyStep`
Perform one iteration of trek separation identification.

Description

A function that does one step through all the nodes in a mixed graph and tries to identify new edge coefficients using trek-separation as described in Weihs, Robeva, Robinson, et al. (2017).

Usage

```
trekSeparationIdentifyStep(
  mixedGraph,
  unsolvedParents,
  solvedParents,
  identifier,
  maxSubsetSize = 3
)
```

Arguments

`mixedGraph` a `MixedGraph` object representing the mixed graph.

`unsolvedParents` a list whose ith index is a vector of all the parents j of i in G which for which the edge j->i is not yet known to be generically identifiable.

`solvedParents` the complement of `unsolvedParents`, a list whose ith index is a vector of all parents j of i for which the edge i->j is known to be generically identifiable (perhaps by other algorithms).

`identifier` an identification function that must produce the identifications corresponding to those in solved parents. That is `identifier` should be a function taking a single argument Sigma (any generically generated covariance matrix corresponding to the mixed graph) and returns a list with two named arguments

Lambda denote the number of nodes in `mixedGraph` as n. Then Lambda is an nxn matrix whose i,jth entry

1. equals 0 if i is not a parent of j,
2. equals NA if i is a parent of j but `identifier` cannot identify it generically,
3. equals the (generically) unique value corresponding to the weight along the edge i->j that was used to produce Sigma.

Omega just as Lambda but for the bidirected edges in the mixed graph

such that if j is in `solvedParents[[i]]` we must have that Lambda[j,i] is not NA.

`maxSubsetSize` a positive integer which controls the maximum subset size considered in the trek-separation identification algorithm. Making this parameter smaller means the algorithm will be faster but less exhaustive (and hence less powerful).

Value

see the return of `htcIdentifyStep`.

`trFrom` *Trek reachable nodes.*

Description

Gets all nodes that are trek reachable from a collection of nodes.

Usage

```
trFrom(this, nodes, ...)

## S3 method for class 'LatentDigraphFixedOrder'
trFrom(
  this,
  nodes,
  avoidLeftNodes = integer(0),
  avoidRightNodes = integer(0),
  includeObserved = T,
  includeLatents = T,
  ...
)

## S3 method for class 'LatentDigraph'
trFrom(
  this,
  nodes,
  avoidLeftNodes = integer(0),
  avoidRightNodes = integer(0),
  includeObserved = T,
  includeLatents = T,
  ...
)

## S3 method for class 'MixedGraph'
trFrom(
  this,
```

```
    nodes,
    avoidLeftNodes = integer(0),
    avoidRightNodes = integer(0),
    ...
  )
```

Arguments

this	the graph object
nodes	the nodes from which to find trek-reachable nodes.
...	ignored.
avoidLeftNodes	a collection of nodes to avoid on the left
avoidRightNodes	a collection of nodes to avoid on the right
includeObserved	if TRUE includes observed nodes in the returned set.
includeLatents	if TRUE includes latent nodes in the returned set.

---

updateEdgeCapacities *Update edge capacities.*

Description

Update edge capacities.

Usage

```
updateEdgeCapacities(this, edges, newCaps)

## S3 method for class 'FlowGraph'
updateEdgeCapacities(this, edges, newCaps)
```

Arguments

this	the flow graph object
edges	the vertices to update (as a 2xr matrix with ith column corresponding to the edge edges[1,i]->edges[2,i].
newCaps	the new capacities for the edges

updateVertexCapacities *Update vertex capacities.*

Description

Update vertex capacities.

Usage

```
updateVertexCapacities(this, vertices, newCaps)

## S3 method for class 'FlowGraph'
updateVertexCapacities(this, vertices, newCaps)
```

Arguments

`this`	the flow graph object
`vertices`	the vertices to update.
`newCaps`	the new capacities for the vertices.

validateLatentNodesAreSources *A helper function to validate that latent nodes in a LatentDigraph are sources.*

Description

Produces an error if not all latent nodes are sources.

Usage

```
validateLatentNodesAreSources(graph)
```

Arguments

`graph`	the LatentDigraph

validateMatrices *A helper function to validate input matrices.*

Description

This helper function validates that the two input matrices, L and O, are of the appropriate form to be interpreted by the other functions. In particular they should be square matrices of 1's and 0's with all 0's along their diagonals. We do not require O to be symmetric here.

Usage

```
validateMatrices(L, O)
```

Arguments

`L`	See above description.
`O`	See above description.

Value

This function has no return value.

validateMatrix *A helper function to validate an input matrix.*

Description

This helper function validates that an input matrix, L, is of the the appropriate form to be interpreted by the other functions. In particular it should be square matrix of 1's and 0's with all 0's along its diagonal. If any of the above conditions is not met, this function will throw an error.

Usage

```
validateMatrix(L)
```

Arguments

`L`	See above description.

Value

No return value

`validateNodes` *A helper function to validate if input nodes are valid.*

Description

Produces an error if outside bounds.

Usage

```
validateNodes(nodes, numNodes)
```

Arguments

`nodes`	the input nodes, expected to be from the collection 1:(number of nodes in the graph)
`numNodes`	the number of observed nodes in the graph.

`validateVarArgsEmpty` *A helper function to validate that there are no variable arguments*

Description

Produces an error if there are variable arguments.

Usage

```
validateVarArgsEmpty(...)
```

Arguments

`...`	the variable arguments

`ZUTA` *Check the Zero Upper Triangular Assumption*

Description

Check the Zero Upper Triangular Assumption

Usage

```
ZUTA(lambda)
```

Arguments

`lambda`	adjacency matrix with number of cols = number of latent nodes, number of rows = number of observed nodes

Value

a Boolean, whether the graph fulfills ZUTA

References

Sturma, N., Kranzlmüller, M., Portakal, I., and Drton, M. (2025) Matching Criterion for Identifiability in Sparse Factor Analysis. arXiv:2502.02986

Index