

Package ‘SEMID’

October 15, 2017

Type Package

Title Identifiability of Linear Structural Equation Models

Version 0.3.1

Date 2017-10-14

Maintainer Luca Weihs <luca@uw.edu>

Description Provides routines to check identifiability or non-identifiability of linear structural equation models as described in Drton, Foygel, and Sullivan (2011) <DOI:10.1214/10-AOS859>, Foygel, Draisma, and Drton (2012) <DOI:10.1214/12-AOS1012>, and other works. The routines are based on the graphical representation of structural equation models by a path diagram/mixed graph.

License GPL (>= 2)

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)

Suggests testthat

NeedsCompilation no

Repository CRAN

RoxygenNote 6.0.1.9000

Author Rina Foygel Barber [aut],
Mathias Drton [aut],
Luca Weihs [cre, aut]

Date/Publication 2017-10-14 23:02:10 UTC

R topics documented:

SEMID-package	3
ancestors	5
ancestralID	6
ancestralIdentifyStep	6

createAncestralIdentifier	7
createEdgewiseIdentifier	8
createHalfTrekFlowGraph	9
createHtcIdentifier	9
createHtrGraph	10
createIdentifierBaseCase	11
createSimpleBiDirIdentifier	11
createTrekFlowGraph	12
createTrekSeparationIdentifier	12
createTrGraph	13
descendants	14
edgewiseID	14
edgewiseIdentifyStep	15
edgewiseTSID	16
flowBetween	17
FlowGraph	17
generalGenericID	18
getAncestors	19
getDescendants	19
getHalfTrekSystem	20
getMaxFlow	20
getMixedCompForNode	21
getParents	22
getSiblings	22
getTrekSystem	23
graphID	23
graphID.ancestralID	25
graphID.decompose	26
graphID.genericID	27
graphID.globalID	27
graphID.htcID	28
graphID.main	29
graphID.nonHtcID	30
htcID	30
htcIdentifyStep	31
htr	32
htrFrom	33
inducedSubgraph	33
isSibling	34
L	34
MixedGraph	35
MixedGraphFixedOrder	36
mixedGraphHasSimpleNumbering	36
nodes	37
numNodes	37
O	38
parents	38
plot.MixedGraph	39

plotMixedGraph	39
print.GenericIDResult	40
print.SEMIDResult	40
semID	41
siblings	42
stronglyConnectedComponent	43
stronglyConnectedComponents	43
subsetsOfSize	44
tianComponent	44
tianDecompose	45
tianIdentifier	45
tianSigmaForComponent	46
toEx	46
toIn	47
trekSeparationIdentifyStep	47
trFrom	48
updateEdgeCapacities	49
updateVertexCapacities	49
validateMatrices	50

Index	51
--------------	-----------

SEMID-package

SEMID package documentation.

Description

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

Details

The only function you're likely to need from **SEMID** is **semID**. 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))
```



```

                                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)

```

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 'MixedGraphFixedOrder'
ancestors(this, nodes)

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

```

Arguments

<code>this</code>	the mixed graph object
<code>nodes</code>	the nodes from which to find all ancestors

ancestralID	<i>Determines which edges in a mixed graph are ancestralID-identifiable</i>
-------------	---

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	<i>Perform one iteration of ancestral identification.</i>
-----------------------	---

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 i th index is a vector of all the parents j of i in G which for which the edge $j \rightarrow i$ is not yet known to be generically identifiable.
- solvedParents** the complement of `unsolvedParents`, a list whose i th index is a vector of all parents j of i for which the edge $i \rightarrow j$ is known to be generically identifiable (perhaps by other algorithms).
- identifier** an identification function that must produce the identifications corresponding to those in `solvedParents`. 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 $n \times n$ matrix whose i, j th 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 $i \rightarrow j$ 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

References

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

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 c-component containing node in the subgraph induced by ancestralSubset. See tianDecompose for how such c-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

createHalfTrekFlowGraph

Helper function to create a flow graph.

Description

Helper function to create a flow graph.

Usage

```
createHalfTrekFlowGraph(this)
```

```
## S3 method for class 'MixedGraphFixedOrder'
createHalfTrekFlowGraph(this)
```

Arguments

this the mixed graph object

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.

createHtrGraph	<i>Helper function to create a graph encoding htr relationships.</i>
----------------	--

Description

Helper function to create a graph encoding htr relationships.

Usage

```
createHtrGraph(this)

## S3 method for class 'MixedGraphFixedOrder'
createHtrGraph(this)
```

Arguments

this	the mixed graph object
------	------------------------

`createIdentifierBaseCase`*Create an identifier base case*

Description

Identifiers are functions that take as input a covariance matrix Σ 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: Λ - a matrix equal to L but with NA values instead of 1s, Ω - a matrix equal to O but with NA values instead of 1s. When building more complex identifiers these NAs will be replaced by the value that can be identified from Σ .

`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 'MixedGraphFixedOrder'
createTrekFlowGraph(this)
```

Arguments

this the mixed graph object

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	<i>Helper function to create a graph encoding trek reachable relationships.</i>
---------------	---

Description

Helper function to create a graph encoding trek reachable relationships.

Usage

```
createTrGraph(this)

## S3 method for class 'MixedGraphFixedOrder'
createTrGraph(this)
```

Arguments

this	the mixed graph object
------	------------------------

descendants	<i>Get descendants of a node</i>
-------------	----------------------------------

Description

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

Usage

```
descendants(this, node)

## S3 method for class 'MixedGraphFixedOrder'
descendants(this, node)

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

Arguments

this	the mixed graph object
node	the node from which to get the descendants.

edgewiseID	<i>Determines which edges in a mixed graph are edgewiseID-identifiable</i>
------------	--

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 *i*th 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 *i*th 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 *n*×*n* matrix whose *i,j*th 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	<i>Determines which edges in a mixed graph are edgewiseID+TS identifiable</i>
--------------	---

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](#).

flowBetween	<i>Flow from one set of nodes to another.</i>
-------------	---

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	<i>Construct FlowGraph object</i>
-----------	-----------------------------------

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 are in `idStepFunctions`

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 *i*th 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 `solvedSiblings` 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	<i>Determines if a half-trek system exists in the mixed graph.</i>
-------------------	--

Description

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

Usage

```
getHalfTrekSystem(this, fromNodes, toNodes)

## S3 method for class 'MixedGraphFixedOrder'
getHalfTrekSystem(this, fromNodes, toNodes)

## S3 method for class 'MixedGraph'
getHalfTrekSystem(this, fromNodes, toNodes)
```

Arguments

this	the mixed graph object
fromNodes	the nodes from which the half-trek system should start. If $\text{length}(\text{fromNodes}) > \text{length}(\text{toNodes})$ will find if there exists any half-trek system from any subset of fromNodes of size $\text{length}(\text{toNodes})$ to toNodes.
toNodes	the nodes where the half-trek system should end.

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	<i>Size of largest HT system Y satisfying the HTC for a node v except perhaps having $\text{getParents}(v) < Y$.</i>
------------	--

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, 0, 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 G_A be the subgraph of H on the nodes A. This function returns the mixed component of G_A 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 could 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.

getParents	<i>Get getParents of nodes in a graph.</i>
------------	--

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	<i>Get getSiblings of nodes in a graph.</i>
-------------	---

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	<i>Determines if a trek system exists in the mixed graph.</i>
---------------	---

Description

Determines if a trek system exists in the mixed graph.

Usage

```
getTrekSystem(this, fromNodes, toNodes, avoidEdgesOnRight)
```

```
## S3 method for class 'MixedGraphFixedOrder'
getTrekSystem(this, fromNodes, toNodes,
  avoidEdgesOnRight = NULL)
```

```
## S3 method for class 'MixedGraph'
getTrekSystem(this, fromNodes, toNodes,
  avoidEdgesOnRight = NULL)
```

Arguments

this	the mixed graph object
fromNodes	the start nodes
toNodes	the end nodes
avoidEdgesOnRight	a collection of edges in the graph that should not be used on any right hand side of any trek in the trek system.

graphID	<i>Identifiability of linear structural equation models.</i>
---------	--

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

Arguments

<code>L</code>	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 $L[i,i]=1$.
<code>O</code>	Adjacency matrix for the bidirected part of the path diagram/mixed graph. Edges are encoded as for the <code>L</code> 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 $O[j,i]=1$.
<code>output.type</code>	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.
<code>file.name</code>	A character string naming the output file.
<code>decomp.if.acyclic</code>	A logical value indicating whether an input graph that is acyclic is to be decomposed before applying identifiability criteria.
<code>test.globalID</code>	A logical value indicating whether or not global identifiability is checked.
<code>test.genericID</code>	A logical value indicating whether or not a sufficient condition for generic identifiability is checked.
<code>test.nonID</code>	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 Sullivan, 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	<i>Determine generic identifiability by Tian Decomposition and HTC</i>
-------------------	--

Description

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

Usage

```
graphID.decompose(L, 0, 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 $L[i,i]=1$.
0	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 occurred, false if not.

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

Description

If 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, 0)

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 $L[i,i]=1$.
- 0 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.globalID *Check for global identifiability of a mixed graph.*

Description

Checks for the global identifiability of a mixed graph using techniques presented in Drton, Foygel, Sullivant (2011).

Usage

graphID.globalID(L, 0)

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 was globally identifiable, FALSE otherwise.

References

Drton, Mathias; Foygel, Rina; Sullivant, Seth. Global identifiability of linear structural equation models. *Ann. Statist.* 39 (2011), no. 2, 865–886.

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	<i>Helper function to handle a graph component.</i>
--------------	---

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, 0)
```

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 $L[i,i]=1$.
- 0** 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.

<code>htcIdentifyStep</code>	<i>Perform one iteration of HTC identification.</i>
------------------------------	---

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 *i*th 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 *i*th 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 *n*x*n* matrix whose *i,j*th 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 $i \rightarrow j$ 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 $r \times 2$ matrix where r is the number of edges that were identified by this function call and `identifiedEdges[i,1] -> identifiedEdges[i,2]` was the i th 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	<i>Half trek reachable nodes.</i>
---------	-----------------------------------

Description

Half trek reachable nodes.

Usage

```
htrFrom(this, node)

## S3 method for class 'MixedGraphFixedOrder'
htrFrom(this, node)

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

Arguments

this	the mixed graph object
node	the node from which to get all half-trek reachable nodes.

Value

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

inducedSubgraph	<i>Get the induced subgraph on a collection of nodes</i>
-----------------	--

Description

Get the induced subgraph on a collection of nodes

Usage

```
inducedSubgraph(this, nodes)

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

Arguments

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

isSibling	<i>Are two nodes siblings?</i>
-----------	--------------------------------

Description

Are two nodes siblings?

Usage

```
isSibling(this, node1, node2)

## S3 method for class 'MixedGraphFixedOrder'
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

Value

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

L	<i>Get adjacency matrix for directed part.</i>
---	--

Description

Get adjacency matrix for directed part.

Usage

```
L(this)

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

Arguments

this	the mixed graph object
------	------------------------

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 = 1: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

MixedGraphFixedOrder *Construct MixedGraphFixedOrder object*

Description

Creates an object representing a mixed graph.

Usage

```
MixedGraphFixedOrder(L = matrix(0,1,1), O = matrix(0,1,1))
```

Arguments

L see [graphID](#) for the appropriate form of L.
O as for L.

Value

An object representing the MixedGraphFixedOrder

mixedGraphHasSimpleNumbering
Checks 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	<i>Get all nodes in the graph.</i>
-------	------------------------------------

Description

Get all nodes in the graph.

Usage

```
nodes(this)

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

Arguments

this the mixed graph object

numNodes	<i>Number of nodes in the graph.</i>
----------	--------------------------------------

Description

Number of nodes in the graph.

Usage

```
numNodes(this)

## S3 method for class 'MixedGraphFixedOrder'
numNodes(this)

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

Arguments

this the mixed graph object

0	<i>Get adjacency matrix for bidirected part.</i>
---	--

Description

Get adjacency matrix for bidirected part.

Usage

```
O(this)

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

Arguments

this the mixed graph object

parents	<i>All parents a collection of nodes.</i>
---------	---

Description

All parents a collection of nodes.

Usage

```
parents(this, nodes)

## S3 method for class 'MixedGraphFixedOrder'
parents(this, nodes)

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

Arguments

this the mixed graph object.
nodes nodes the nodes of which to find the parents.

Value

a vector of parents of the nodes.

plot.MixedGraph	<i>Plots the mixed graph</i>
-----------------	------------------------------

Description

Plots the mixed graph

Usage

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

Arguments

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

plotMixedGraph	<i>Plot a mixed graph</i>
----------------	---------------------------

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.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.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.

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 [graphID.globalID](#) function.

testGenericNonID TRUE or 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 'MixedGraphFixedOrder'
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.

Value

a vector of all of the siblings.

`stronglyConnectedComponent`
Strongly connected component

Description

Get the strongly connected component for a node *i* in the directed part of the graph.

Usage

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

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

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

Arguments

<code>this</code>	the mixed graph object
<code>node</code>	the node for which to get the strongly connected component.

`stronglyConnectedComponents`
Strongly connected components

Description

Get the strongly connected components of a graph

Usage

```
stronglyConnectedComponents(this)
```

```
## S3 method for class 'MixedGraphFixedOrder'  
stronglyConnectedComponents(this)
```

Arguments

<code>this</code>	the mixed graph object
-------------------	------------------------

subsetsOfSize	<i>Returns all subsets of a certain size</i>
---------------	--

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	<i>Returns the Tian c-component of a node</i>
---------------	---

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	<i>Performs the tian decomposition on the mixed graph</i>
---------------	---

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	<i>Identifies components in a tian decomposition</i>
----------------	--

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 Σ of special subgraphs of G called c -components. This function takes the covariance matrix Σ corresponding to G and a collection of node sets which specify the c -component, and returns the Σ corresponding to the c -component.

Usage

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

Arguments

<code>Sigma</code>	the covariance matrix for the mixed graph G
<code>internal</code>	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).
<code>incoming</code>	the parents of vertices in <code>internal</code> that are not in the set <code>internal</code> themselves
<code>topOrder</code>	a topological ordering of $c(\text{internal}, \text{incoming})$ with respect to the graph G . For vertices in a strongly connected component the ordering is allowed to be arbitrary.

Value

the new Σ corresponding to the c -component

<code>toEx</code>	<i>Transforms a vector of node indices in the internal rep. into external numbering</i>
-------------------	---

Description

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

Usage

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

```
## S3 method for class 'MixedGraph'
```

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

Arguments

this	the mixed graph object
nodes	the nodes to transform

toIn	<i>Transforms a vector of given node indices into their internal numbering</i>
------	--

Description

Transforms a vector of given node indices into their internal numbering

Usage

```
toIn(this, nodes)

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

Arguments

this	the mixed graph object
nodes	the nodes to transform

trekSeparationIdentifyStep	<i>Perform one iteration of trek separation identification.</i>
----------------------------	---

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 <i>i</i> th index is a vector of all the parents <i>j</i> of <i>i</i> in <i>G</i> which for which the edge <i>j</i> -> <i>i</i> is not yet known to be generically identifiable.
solvedParents	the complement of <code>unsolvedParents</code> , a list whose <i>i</i> th index is a vector of all parents <i>j</i> of <i>i</i> for which the edge <i>i</i> -> <i>j</i> is known to be generically identifiable (perhaps by other algorithms).
identifier	an identification function that must produce the identifications corresponding to those in <code>solvedParents</code> . That is <code>identifier</code> should be a function taking a single argument <code>Sigma</code> (any generically generated covariance matrix corresponding to the mixed graph) and returns a list with two named arguments <p>Lambda denote the number of nodes in <code>mixedGraph</code> as <i>n</i>. Then <code>Lambda</code> is an <i>n</i>x<i>n</i> matrix whose <i>i,j</i>th entry</p> <ol style="list-style-type: none"> 1. equals 0 if <i>i</i> is not a parent of <i>j</i>, 2. equals NA if <i>i</i> is a parent of <i>j</i> but <code>identifier</code> cannot identify it generically, 3. equals the (generically) unique value corresponding to the weight along the edge <i>i</i>-><i>j</i> that was used to produce <code>Sigma</code>. <p>Omega just as <code>Lambda</code> but for the bidirected edges in the mixed graph such that if <i>j</i> is in <code>solvedParents[[i]]</code> we must have that <code>Lambda[j,i]</code> is not NA.</p>
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	<i>Trek reachable nodes.</i>
--------	------------------------------

Description

Like [htrFrom](#) but for the nodes that are trek-reachable from a node

Usage

```
trFrom(this, node)

## S3 method for class 'MixedGraphFixedOrder'
trFrom(this, node)

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


Arguments

this	the mixed graph object
node	the node from which to find trek-reachable nodes.

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 row corresponding to the edge edges[i,1]->edges[i,2].
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.

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.

Index

ancestors, 5
ancestralID, 6
ancestralIdentifyStep, 6

class, 18, 41
createAncestralIdentifier, 7
createEdgewiseIdentifier, 8
createHalfTrekFlowGraph, 9
createHtcIdentifier, 9
createHtrGraph, 10
createIdentifierBaseCase, 11
createSimpleBiDirIdentifier, 11
createTrekFlowGraph, 12
createTrekSeparationIdentifier, 12
createTrGraph, 13

descendants, 14

edgewiseID, 14
edgewiseIdentifyStep, 8, 15
edgewiseTSID, 16

flowBetween, 17
FlowGraph, 17

generalGenericID, 6, 15, 16, 18, 31, 40, 41
getAncestors, 19
getDescendants, 19
getHalfTrekSystem, 20
getMaxFlow, 20
getMixedCompForNode, 21
getParents, 22
getSiblings, 22
getTrekSystem, 23
graphID, 23, 35, 36
graphID.ancestralID, 6, 25
graphID.decompose, 26
graphID.genericID, 27
graphID.globalID, 27, 41
graphID.htcID, 28, 30
graphID.main, 29

graphID.nonHtcID, 30, 41

htcID, 30
htcIdentifyStep, 9, 16, 18, 19, 31, 41, 48
htr, 32
htrFrom, 33, 48

inducedSubgraph, 33
invisible, 40
isSibling, 34

L, 34

MixedGraph, 6, 7, 14–16, 18, 31, 35, 41, 48
MixedGraphFixedOrder, 36
mixedGraphHasSimpleNumbering, 36

nodes, 37
numNodes, 37

0, 38

parents, 38
plot.MixedGraph, 39
plotMixedGraph, 39
print.GenericIDResult, 40
print.SEMIDResult, 40

SEMID (SEMID-package), 3
semID, 3, 23, 40, 41
SEMID-package, 3
siblings, 42
stronglyConnectedComponent, 43
stronglyConnectedComponents, 43
subsetsOfSize, 44

tianComponent, 44
tianDecompose, 8, 45, 45
tianIdentifier, 45
tianSigmaForComponent, 46
toEx, 46

toIn, [47](#)
trekSeparationIdentifyStep, [12](#), [47](#)
trFrom, [48](#)

updateEdgeCapacities, [49](#)
updateVertexCapacities, [49](#)

validateMatrices, [50](#)