Package 'dagmc'

August 31, 2023

Title MCMC sampler for Directed Acyclic Graphs

Version 0.0.1

Description What the package does (one paragraph).

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Imports magrittr,
bnlearn,
igraph,
doParallel,
parallel,
foreach,
arrangements,
graphics,
dplyr,
rlang,
fastmatch

Suggests rmarkdown,
knitr,
testthat (>= 3.0.0),
gtools,
gRain,
gRbase,
ggplot2

Config/testthat/edition 3

VignetteBuilder knitr

R topics documented:

BNLearnScorer *BNLearnScorer*

Description

A thin wrapper on the bnlearn::score function.

Usage

```
BNLearnScorer(node, parents, ...)
```

Arguments

node	Name of node to score.
parents	The parents of node.
...	The ellipsis is used to pass other parameters to the scorer.

Examples

```
data <- bnlearn::learning.test
BNLearnScorer('A', c('B', 'C'), data = data)
BNLearnScorer('A', c(), data = data)
BNLearnScorer('A', vector(), data = data)
BNLearnScorer('A', NULL, data = data)
BNLearnScorer('A', c('B', 'C'), data = data, type = "bde", iss = 100)
BNLearnScorer('A', c('B', 'C'), data = data, type = "bde", iss = 1)
```

CachedScorer *This builds the score cache. It can be used for problems where the score only changes as a function of (node, parents).*

Description

This builds the score cache. It can be used for problems where the score only changes as a function of (node, parents).

Usage

```
CachedScorer(scorer, max_size = NULL)
```

Arguments

scorer	A scorer.
max_size	Not implemented. Maximum number of scores to store in the cache. If the total number of combinations is greater than this number then the cache follows a least recently used replacement policy.

Examples

```
scorer <- CreateScorer(data = bnlearn::learning.test)
cached_scorer <- CachedScorer(scorer)
cached_scorer('A', c('B', 'C'))
```

`CalculateAcceptanceRates` *Calculate acceptance rates per proposal.*

Description

This makes the assumption that the proposal has saved a variable "proposal_used" and mcmc has saved a variable 'accept'.

Usage

```
CalculateAcceptanceRates(chains, group_by = NULL)
```

Arguments

`chains`	MCMC chains.
`group_by`	Vector of strings that are in c("chain", "proposal_used"). Default is NULL which will return the acceptance rates marginalised over chains and the proposal used.

Value

Summary of acceptance rates per grouping.

`CalculateNodeMoveNeighbourhood` *Calculate neighbourhood for node move.*

Description

Calculate neighbourhood for node move.

Usage

```
CalculateNodeMoveNeighbourhood(partitioned_nodes)
```

Arguments

`partitioned_nodes`	Labelled partition.

`CalculateSplitJoinNeighbourhood` *Calculate neighbourhood for the split or join proposal.*

Description

The number of split combinations prescribed by KP15 is ambiguous when a partition element has only 1 node. A split for a partition element with 1 node results in a proposal to stay still, as such I remove that proposal.

Usage

```
CalculateSplitJoinNeighbourhood(partitioned_nodes)
```

Arguments

`partitioned_nodes` Labelled partition.

`CalculateStayStillNeighbourhood` *Calculate neighbourhood for staying still.*

Description

Calculate neighbourhood for staying still.

Usage

```
CalculateStayStillNeighbourhood(partitioned_nodes)
```

Arguments

`partitioned_nodes` A labelled partition.

`CalculateSwapAdjacentNodeNeighbourhood` *Calculate neighbourhood for swapping nodes.*

Description

Calculate neighbourhood for swapping nodes.

Usage

```
CalculateSwapAdjacentNodeNeighbourhood(partitioned_nodes)
```

Arguments

`partitioned_nodes` Labelled partition.

`CalculateSwapNodeNeighbourhood` *Calculate neighbourhood for swapping nodes.*

Description

Calculate neighbourhood for swapping nodes.

Usage

```
CalculateSwapNodeNeighbourhood(partitioned_nodes)
```

Arguments

`partitioned_nodes` Labelled partition.

`CheckBlacklistObeyed` *Check blacklist obeyed.*

Description

If an edge between two nodes is blacklisted in Partition MCMC the adjacent partition element cannot be the only direct node for it's blacklisted child.

Usage

```
CheckBlacklistObeyed(partitioned_nodes, blacklist = NULL, nodes = NULL)
```

Arguments

partitioned_nodes	Labelled partition.
blacklist	A data.frame of (parent, child) pairs representing edges that cannot be in the DAG.
nodes	A vector of node names to check. Default is to check all child nodes in the blacklist.

CheckWhitelistObeyed *Check whitelist is obeyed.*

Description

Check whitelist is obeyed.

Usage

```
CheckWhitelistObeyed(partitioned_nodes, whitelist = NULL, nodes = NULL)
```

Arguments

partitioned_nodes	Labelled partition.
whitelist	A data.frame of (parent, child) pairs representing edges that must be in the DAG.
nodes	A vector of node names to check. Default is to check all child nodes in the whitelist.

CollectUniqueObjects *Collect unique state objects.*

Description

Get the unique set of states and DAGs along with their log score.

Usage

```
CollectUniqueObjects(chain)
```

Arguments

chain	A chain that includes a DAG per sample.

Details

This gets the unique set of states and DAGs which are referred to as objects ($\mathcal{O}$). Then estimates the log of the normalisation constant assuming $\tilde{Z}_{\mathcal{O}} = \Sigma_s^S p(\mathcal{O}_s)p(D|\mathcal{O}_s)$ where $\{\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, ..., \mathcal{O}_S\}$ is the set of unique objects in the chain. This assumes that you have captured the most probable objects, such that $\tilde{Z}_{\mathcal{O}}$ is approximately equal to the true evidence $Z = \Sigma_{G \in \mathcal{G}} p(G)p(D|G)$ where you sum across all possible DAGs ($\mathcal{G}$). This also makes the assumption that the exponential of the score is proportional to the posterior probability, such that

$$p(G|D) \propto p(G)p(D|G) = \prod_i \exp(\text{score}(X_i, \text{Pa}_G(X_i)|D))$$

where $\text{Pa}_G(X_i)$ is the parents set for node X_i.

We calculate the estimator using both the states (e.g., labelled partitions) and DAGs. The estimator using the labelled partitions is more accurate as it includes the sum over a greater number of DAGs. However, they should be approximately the same value. If they are not, then you probably haven't sampled enough DAGs from your states.

After the normalisation constant has been estimated we then estimate the log probability of each graph as,

$$\log(p(\mathcal{O}|D)) = \log(p(\mathcal{O})p(D|\mathcal{O})) - \log(\tilde{Z}_{\mathcal{O}})$$

.

Value

dag_collection: A list with entries:

- states: List of unique states.
- log_evidence_states: Numeric value representing the evidence calculated from the states.
- log_state_scores: Vector with the log scores for each state.
- dags: List of unique DAGs.
- dag_scores: Vector with the log scores for each DAG.
- log_norm_dag_scores: Vector of normalised dag scores.
- log_evidence_dags: Numeric value representing the evidence calculated from the DAGs.

`CreateScorer` *Scorer constructor.*

Description

Scorer constructor.

Usage

```
CreateScorer(
  scorer = BNLearnScorer,
  ...,
  max_parents = Inf,
  blacklist = NULL,
  whitelist = NULL,
  cache = FALSE
)
```

Arguments

scorer	A scorer function that takes (node, parents) as parameters. Default is BNLearnScorer.
...	Parameters to pass to scorer.
max_parents	The maximum number of allowed parents. Default is infinite.
blacklist	A boolean matrix of (parent, child) pairs where TRUE represents edges that cannot be in the DAG. Default is NULL which represents no blacklisting.
whitelist	A boolean matrix of (parent, child) pairs where TRUE represents edges that must be in the DAG. Default is NULL which represents no whitelisting.
cache	A boolean to indicate whether to build the cache. The cache only works for problems where the scorer only varies as a function of (node, parents). Default is FALSE.

Examples

```
scorer <- CreateScorer(data = bnlearn::asia)
```

DefaultProposal *Default proposal constructor.*

Description

Default proposal constructor.

Usage

```
DefaultProposal(p = c(0.33, 0.33, 0.165, 0.165, 0.01), verbose = TRUE)
```

Arguments

p	Probability for each proposal in the order (split_join, node_move, swap_node, swap_adjacent, stay_still).
verbose	Boolean flag to record proposal used.

FindChangedNodes *Find nodes with changed parent combinations between different labelled partitions.*

Description

TODO: This is quite slow. From the proposal we should be able to determine the nodes that need to be rescored rather than finding them using this function.

Usage

```
FindChangedNodes(old_partitioned_nodes, new_partitioned_nodes, scorer)
```

Arguments

old_partitioned_nodes	Labelled partition.
new_partitioned_nodes	Labelled partition.
scorer	Scorer object.

Value

Vector of changed nodes.

Examples

```
scorer = CreateScorer()

old_dag <- UniformlySampleDAG(LETTERS[1:5])
old_partitioned_nodes <- GetPartitionedNodesFromAdjacencyMatrix(old_dag)

new_dag <- UniformlySampleDAG(LETTERS[1:5])
new_partitioned_nodes <- GetPartitionedNodesFromAdjacencyMatrix(new_dag)

changed_nodes <- FindChangedNodes(old_partitioned_nodes, new_partitioned_nodes, scorer)
```

FlattenChains *Flatten list of chains.*

Description

Flatten list of chains.

Usage

```
FlattenChains(chains)
```

Arguments

chains	MCMC chains.

GetEmptyDAG *Get an empty DAG given a set of nodes.*

Description

Get an empty DAG given a set of nodes.

Usage

```
GetEmptyDAG(nodes)
```

Arguments

nodes	A vector of node names.

Value

An adjacency matrix with elements designated as (parent, child).

GetLowestPairwiseScoringEdges *Preprocessing for blacklisting. Get the lowest pairwise scoring edges.*

Description

Get the lowest c pairwise scoring edges represented as a blacklist matrix. This blacklisting procedure is motivated by Koller & Friedman (2003).

Usage

```
GetLowestPairwiseScoringEdges(scorer, n_retain)
```

Arguments

scorer	A scorer object.
n_retain	An integer representing the number of edges to retain.

Value

A boolean matrix of (parent, child) pairs for blacklisting.

References

1. Koller D, Friedman N. Being Bayesian about network structure. A Bayesian approach to structure discovery in Bayesian networks. Mach Learn. 2003;50(1):95–125.

GetNodePartition *Get a node's partition element number.*

Description

Get a node's partition element number.

Usage

```
GetNodePartition(partitioned_nodes, node)
```

Arguments

partitioned_nodes	Labelled partition.
node	Node name.

Value

Node's partition element number.

`GetNumberOfPartitions` *Get number of partitions.*

Description

Calculate the number of partitions for a given labelled partition. This is 'm' in Kuipers & Moffa (2015).

Usage

```
GetNumberOfPartitions(partitioned_nodes)
```

Arguments

`partitioned_nodes`
: Labelled partition.

`GetOrderedPartition` *Get ordered labelled partition.*

Description

Calculate the ordered partition. Denoted as lamba in Kuipers & Moffa (2015).

Usage

```
GetOrderedPartition(partitioned_nodes)
```

Arguments

`partitioned_nodes`
: Labelled partition.

Value

Ordered partition.

GetParentCombinations *Get parent combinations for a given node.*

Description

Get parent combinations for a given node.

Usage

```
GetParentCombinations(partitioned_nodes, node, scorer)
```

Arguments

partitioned_nodes	Labelled partition.
node	Node name.
scorer	A scorer object.

Value

List of parent combinations.

GetPartitionedNodesFromAdjacencyMatrix *Map DAG to a labelled partition.*

Description

This partitions nodes into levels of outpoints as explained in Section 4.1 of Kuipers & Moffa 2015. This takes an adjacency matrix and returns a data.frame of (partition, node) pairs

Usage

```
GetPartitionedNodesFromAdjacencyMatrix(adjacency)
```

Arguments

adjacency	Adjacency matrix.

Value

Labelled partition for the given adjacency matrix.

GetPartitionNodes *Get nodes in a partition element.*

Description

Get nodes in a partition element.

Usage

```
GetPartitionNodes(partitioned_nodes, elements)
```

Arguments

partitioned_nodes	Labelled partition.
elements	An integer or vector of integers for the partition element number.

GetRestrictedNodes *Get nodes that have restricted parents.*

Description

Get nodes that have restricted parents.

Usage

```
GetRestrictedNodes(list)
```

Arguments

list	A black or white list.

GetRestrictedParents *Get black or white listed parents.*

Description

Get black or white listed parents.

Usage

```
GetRestrictedParents(node, listed = NULL)
```

Arguments

node	The name of the node to get white or black listed parents.
listed	A black or white list.

`LogSumExp` *Log-Sum-Exponential calculation using the trick that limits underflow issues.*

Description

Log-Sum-Exponential calculation using the trick that limits underflow issues.

Usage

```
LogSumExp(x)
```

Arguments

`x`	A vector of numeric.

Value

Log-Sum-Exponential (LSE) of x.

`NodeMove` *Node move proposal.*

Description

Node move proposal.

Usage

```
NodeMove(partitioned_nodes)
```

Arguments

`partitioned_nodes`	Labelled partition.

`OrderPartitionedNodes` *Order partitioned nodes.*

Description

Order partitioned nodes.

Usage

```
OrderPartitionedNodes(partitioned_nodes)
```

Arguments

`partitioned_nodes`	Labelled partition.

Value

Labelled partitioned in descending partition element order.

`PartitionMCMC` *Transition objects. A one step implementation of partition MCMC. This acts as a constructor.*

Description

This is a constructor for a single Partition MCMC step. The function constructs an environment with the proposal and verbose flag. It then returns a function which takes the current_state and a scorer object.

Usage

```
PartitionMCMC(proposal = NULL, verbose = TRUE)
```

Arguments

`proposal`	Proposal function. Default is the DefaultProposal.
`verbose`	Flag to pass mcmc information.

Value

Function that takes the current state and scorer that outputs a new state.

Examples

```
dag <- UniformlySampleDAG(c('A', 'B', 'C', 'D', 'E', 'F'))
partitioned_nodes <- GetPartitionedNodesFromAdjacencyMatrix(dag)

scorer <- list(
  scorer = BNLearnScorer,
  parameters = list(data = bnlearn::learning.test)
  )

current_state <- list(
  state = partitioned_nodes,
  log_score = ScoreLabelledPartition(partitioned_nodes, scorer)
  )

pmcmc <- PartitionMCMC(proposal = PartitionSplitJoin)
pmcmc(current_state, scorer)
```

PartitionSplitJoin *Partition split or join constructor.*

Description

Partition split or join constructor.

Usage

```
PartitionSplitJoin(partitioned_nodes)
```

Arguments

partitioned_nodes	Labelled partition.

PlotScoreTrace *Plot the score trace.*

Description

Plot the score trace.

Usage

```
PlotScoreTrace(
  chains,
  attribute = "log_score",
  n_burnin = 0,
  same_plot = TRUE,
  col = NULL,
  ...
)
```

Arguments

chains	MCMC chains.
attribute	Name of attribute to plot. Default is "log_score".
n_burnin	Number of steps to remove as burnin.
same_plot	Whether to plot on the same figure or on multiple figures.
col	A string representing a color for a single chain or a vector of strings to cycle through for multiple chains.
...	Extra parameters to pass to the plot and graphics::line functions.

PostProcessChains *Analysis of chains. Equilibrium states.*

Description

This allows you to remove a burnin and thin the chains after processing.

Usage

```
PostProcessChains(chains, n_burnin = 0, n_thin = 1)
```

Arguments

chains	MCMC chains.
n_burnin	Number of steps to remove at the start as a burnin. Default is 0.
n_thin	Number of steps between retained states. Default is 1.

ProposeNodeMove *Propose individual node movement.*

Description

This proposes that a single node selected uniformly can either:

1. Move to any current partition.
2. Move to any gap between or at the ends of the partitions.

Any of these moves are possible and are selected uniformly with two exceptions:

1. The selected node cannot move into adjacent gaps if it originated from a single node partition.
2. The selected node cannot move to the immediately higher gap if it originated from a two node partition.

Usage

```
ProposeNodeMove(partitioned_nodes)
```

Arguments

partitioned_nodes
: Labelled partition.

Examples

```
dag <- UniformlySampleDAG(c('A', 'B', 'C', 'D', 'E', 'F'))
partitioned_nodes <- GetPartitionedNodesFromAdjacencyMatrix(dag)
ProposeNodeMove(partitioned_nodes)
```

ProposePartitionSplitJoin

Propose a split or join of two partitions.

Description

This is the 'Basic Move' (i.e. algorithm 1) in Kuipers & Moffa (2015). There is a caveat in that the split proposal for a partition with one element is ambiguous, as a split for such a partition element results in a stay still proposal. Such a proposal has been removed.

Usage

```
ProposePartitionSplitJoin(partitioned_nodes)
```

Arguments

partitioned_nodes
: A labelled partition.

Value

A proposed labelled partition.

Examples

```
dag <- UniformlySampleDAG(c('A', 'B', 'C', 'D', 'E', 'F'))
partitioned_nodes <- GetPartitionedNodesFromAdjacencyMatrix(dag)
ProposePartitionSplitJoin(partitioned_nodes)
```

ProposeStayStill *Propose that the partition stays still.*

Description

Propose that the partition stays still.

Usage

```
ProposeStayStill(partitioned_nodes)
```

Arguments

partitioned_nodes	A labelled partition.

Value

A proposed labelled partition.

Examples

```
dag <- UniformlySampleDAG(c('A', 'B', 'C', 'D', 'E', 'F'))
partitioned_nodes <- GetPartitionedNodesFromAdjacencyMatrix(dag)
ProposeStayStill(partitioned_nodes)
```

ProposeSwapAdjacentNode *Propose that two nodes swap partition elements.*

Description

Propose that two nodes swap partition elements.

Usage

```
ProposeSwapAdjacentNode(partitioned_nodes)
```

Arguments

partitioned_nodes	labelled partition.

Value

A proposed labelled partition.

Examples

```
dag <- UniformlySampleDAG(c('A', 'B', 'C', 'D', 'E', 'F'))
partitioned_nodes <- GetPartitionedNodesFromAdjacencyMatrix(dag)
ProposeStayStill(partitioned_nodes)
```

ProposeSwapNode *Propose that two nodes swap partition elements.*

Description

Propose that two nodes swap partition elements.

Usage

```
ProposeSwapNode(partitioned_nodes)
```

Arguments

partitioned_nodes	labelled partition.

Value

A proposed labelled partition.

Examples

```
dag <- UniformlySampleDAG(c('A', 'B', 'C', 'D', 'E', 'F'))
partitioned_nodes <- GetPartitionedNodesFromAdjacencyMatrix(dag)
ProposeStayStill(partitioned_nodes)
```

SampleChain *Sample a single chain.*

Description

Sample a single chain.

Usage

```
SampleChain(n_results, init_state, transition, scorer, n_thin = 1)
```

Arguments

n_results	Number of saved states.
init_state	An initial state that can be passed to transition.
transition	A transition function.
scorer	A scorer object.
n_thin	Number of steps between saved states.

Examples

```
data <- bnlearn::learning.test

dag <- UniformlySampleDAG(colnames(data))
partitioned_nodes <- GetPartitionedNodesFromAdjacencyMatrix(dag)

scorer_1 <- list(
  scorer = BNLearnScorer,
  parameters = list(data = data)
  )

results <- SampleChain(10, partitioned_nodes, PartitionMCMC(), scorer_1)
```

SampleChainDAGs *Sampled DAG from chains.*

Description

Sampled DAG from chains.

Usage

```
SampleChainDAGs(chains, scorer)
```

Arguments

chains	MCMC chains.
scorer	Scorer object.

Value

Chains with sample dags and their corresponding score.

SampleChains *Sample multiple chains in parallel.*

Description

Sample multiple chains in parallel.

Usage

```
SampleChains(
  n_results,
  init_state,
  transition,
  scorer,
  n_thin = 1,
  n_parallel_chains = 2
)
```

Arguments

n_results	Number of saved states per chain.
init_state	An initial state that can be passed to transition. This can be a single state or a list of states for each parallel chain.
transition	A transition function.
scorer	A scorer object.
n_thin	Number of steps between saved states.
n_parallel_chains	Number of chains to run in parallel. Default is 2.

Value

List of results.

Examples

```
data <- bnlearn::learning.test

dag <- UniformlySampleDAG(colnames(data))
partitioned_nodes <- GetPartitionedNodesFromAdjacencyMatrix(dag)

scorer <- list(
  scorer = BNLearnScorer,
  parameters = list(data = data)
  )

results <- SampleChains(10, partitioned_nodes, PartitionMCMC(), scorer)
```

SampleDAGFromLabelledPartition

Sample a DAG from a labelled partition.

Description

Sample a DAG from a labelled partition.

Usage

```
SampleDAGFromLabelledPartition(partitioned_nodes, scorer)
```

Arguments

partitioned_nodes	Labelled partition.
scorer	Scorer object.

Value

A matrix with elements of (parent, child) cells with 1 representing and edge and 0 otherwise.

Examples

```
data <- bnlearn::learning.test

dag <- UniformlySampleDAG(colnames(data))
partitioned_nodes <- GetPartitionedNodesFromAdjacencyMatrix(dag)

scorer <- CreateScorer(data = data)

SampleDAGFromLabelledPartition(partitioned_nodes, scorer = scorer)
```

ScoreDAG *Score DAG.*

Description

Score DAG.

Usage

```
ScoreDAG(dag, scorer)
```

Arguments

dag	Adjacency matrix of (parent, child) entries with 1 denoting an edge and 0 otherwise.
scorer	Scorer object.

Value

Log of DAG score.

ScoreDiff *Calculate the difference in log scores between two labelled partitions.*

Description

Calculate the difference in log scores between two labelled partitions.

Usage

```
ScoreDiff(
  old_partitioned_nodes,
  new_partitioned_nodes,
  scorer,
  rescore_nodes = NULL
)
```

Arguments

old_partitioned_nodes	A labelled partition.
new_partitioned_nodes	A labelled partition.
scorer	A scorer object.
rescore_nodes	Default is NULL which will determine the

Value

Log of score difference between two labelled partitions.

Examples

```
data <- bnlearn::learning.test

old_dag <- UniformlySampleDAG(names(data))
old_partitioned_nodes <- GetPartitionedNodesFromAdjacencyMatrix(old_dag)

new_dag <- UniformlySampleDAG(names(data))
new_partitioned_nodes <- GetPartitionedNodesFromAdjacencyMatrix(new_dag)

scorer <- list(
  scorer = BNLearnScorer,
  parameters = list(data = data)
  )

ScoreDiff(old_partitioned_nodes, new_partitioned_nodes, scorer = scorer)
```

ScoreLabelledPartition

Score labelled partition by adding the log scores for each node.

Description

Score labelled partition by adding the log scores for each node.

Usage

```
ScoreLabelledPartition(partitioned_nodes, scorer)
```

Arguments

partitioned_nodes	Labelled partition.
scorer	Scorer object.

Value

Log of the node score.

Examples

```
data <- bnlearn::learning.test

dag <- UniformlySampleDAG(names(data))
partitioned_nodes <- GetPartitionedNodesFromAdjacencyMatrix(dag)

scorer <- list(
  scorer = BNLearnScorer,
  parameters = list(data = data)
  )

ScoreLabelledPartition(partitioned_nodes, scorer)
```

ScoreNode *Score node by marginalising over parent combinations.*

Description

Score node by marginalising over parent combinations.

Usage

```
ScoreNode(partitioned_nodes, node, scorer)
```

Arguments

partitioned_nodes	Labelled partition.
node	The node name.
scorer	A scorer object.

Value

Log of the node score.

Examples

```
data <- bnlearn::learning.test

dag <- UniformlySampleDAG(names(data))
partitioned_nodes <- GetPartitionedNodesFromAdjacencyMatrix(dag)

scorer <- list(
  scorer = BNLearnScorer,
  parameters = list(data = data)
  )

ScoreNode(partitioned_nodes, 'A', scorer)
```

ScoreTableNode *Calculate score tables for (node, parents) combinations.*

Description

Calculate score tables for (node, parents) combinations.

Usage

```
ScoreTableNode(partitioned_nodes, node, scorer)
```

Arguments

partitioned_nodes	Labelled partition.
node	Name of node.
scorer	Scorer object.

Value

List of log_scores for each combination in parent_combinations.

Examples

```
data <- bnlearn::learning.test

dag <- UniformlySampleDAG(names(data))
partitioned_nodes <- GetPartitionedNodesFromAdjacencyMatrix(dag)

scorer <- list(
  scorer = BNLearnScorer,
  parameters = list(data = data)
  )

ScoreTableNode(partitioned_nodes, 'A', scorer)
```

StayStill *StayStill proposal.*

Description

StayStill proposal.

Usage

```
StayStill(partitioned_nodes)
```

Arguments

partitioned_nodes	Labelled partition.

SwapAdjacentNode *Swap nodes from adjacent partition elements proposal.*

Description

Swap nodes from adjacent partition elements proposal.

Usage

```
SwapAdjacentNode(partitioned_nodes)
```

Arguments

partitioned_nodes	Labelled partition.

SwapNode *Swap node proposal.*

Description

Swap node proposal.

Usage

```
SwapNode(partitioned_nodes)
```

Arguments

partitioned_nodes	Labelled partition.

UniformlySampleDAG *Uniformly sample DAG given a set of nodes.*

Description

Uniformly sample DAG given a set of nodes.

Usage

```
UniformlySampleDAG(nodes)
```

Arguments

nodes	A vector of node names.

Value

Adjacency matrix with elements designated as (parent, child).

Index