

# Package ‘dagmc’

November 9, 2023

**Title** MCMC sampler for Directed Acyclic Graphs

**Version** 0.1.0

## Description

An implementation of the partition MCMC algorithm for Directed Acyclic Graphs. The default implementation should be consistent with BiDAG with some extra functionality allowing you to create your own proposals. The default scoring function uses bnlearn, which means that you have access to all of their scoring options and functionality. You can also build your own scorer if you wish to do so.

**References** Kuipers and Moffa (2017) <doi:10.1080/01621459.2015.1133426>, Scutari (2010) <doi:10.18637/jss.v035.i03>, Suter et al. (2023) <doi:10.18637/jss.v105.i09>.

**License** MIT + file LICENSE

**Encoding** UTF-8

**LazyData** true

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.2.3

**Imports** bnlearn (>= 4.9), igraph, doParallel, parallel, foreach, arrangements, graphics, dplyr, rlang, fastmatch, methods

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

**Config/testthat/edition** 3

**VignetteBuilder** knitr

## R topics documented:

BNLearnScorer . . . . .	3
CachedScorer . . . . .	3
CalculateAcceptanceRates . . . . .	4
CalculateEdgeProbabilities . . . . .	4
CalculateFeatureProbability . . . . .	5
CalculateNodeMoveNeighbourhood . . . . .	5
CalculateSplitJoinNeighbourhood . . . . .	6
CalculateStayStillNeighbourhood . . . . .	6
CalculateSwapAdjacentNodeNeighbourhood . . . . .	7
CalculateSwapNodeNeighbourhood . . . . .	7
CheckBlacklistObeyed . . . . .	7
CheckWhitelistObeyed . . . . .	8
CollectUniqueObjects . . . . .	8

CreateScorer . . . . .	9
DAGtoCPDAG . . . . .	10
DefaultProposal . . . . .	11
FindChangedNodes . . . . .	11
FlattenChains . . . . .	12
GetEmptyDAG . . . . .	12
GetIncrementalScoringEdges . . . . .	13
GetLowestPairwiseScoringEdges . . . . .	13
GetMAP . . . . .	14
GetNodePartition . . . . .	14
GetNumberOfPartitions . . . . .	15
GetOrderedPartition . . . . .	15
GetParentCombinations . . . . .	16
GetParentsKey . . . . .	16
GetPartiallyIncrementalEdges . . . . .	17
GetPartitionedNodesFromAdjacencyMatrix . . . . .	17
GetPartitionNodes . . . . .	18
GetRestrictedNodes . . . . .	18
GetRestrictedParents . . . . .	18
LogSumExp . . . . .	19
NodeMove . . . . .	19
OrderPartitionedNodes . . . . .	20
PartitionMCMC . . . . .	20
PartitionSplitJoin . . . . .	21
PartitiontoDAG . . . . .	21
PlotScoreTrace . . . . .	22
PostProcessChains . . . . .	22
ProposeNodeMove . . . . .	23
ProposePartitionSplitJoin . . . . .	23
ProposeStayStill . . . . .	24
ProposeSwapAdjacentNode . . . . .	24
ProposeSwapNode . . . . .	25
SampleChain . . . . .	26
SampleChains . . . . .	26
SampleDAGFromLabelledPartition . . . . .	27
ScoreDAG . . . . .	28
ScoreDiff . . . . .	29
ScoreLabelledPartition . . . . .	30
ScoreNode . . . . .	30
ScoreTableNode . . . . .	31
StayStill . . . . .	32
SwapAdjacentNode . . . . .	32
SwapNode . . . . .	33
toBNLearn . . . . .	33
toMatrix . . . . .	34
UniformlySampleDAG . . . . .	34
[.dagmc_chain . . . . .	35
[.dagmc_chains . . . . .	35
[[.dagmc_chains . . . . .	36

---

BNLearnScorer

*BNLearnScorer*


---

### Description

A thin wrapper on the `bnlearn::score` function.

### Usage

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

### Arguments

<code>node</code>	Name of node to score.
<code>parents</code>	The parents of node.
<code>...</code>	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, nthreads = 1)
```

### Arguments

<code>scorer</code>	A scorer.
<code>max_size</code>	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.
<code>nthreads</code>	The number of threads used to create the cache.

**Examples**

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

---

CalculateAcceptanceRates

*Calculate acceptance rates.*

---

**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.

---

CalculateEdgeProbabilities

*Calculate marginalised edge probabilities.*

---

**Description**

Calculate the probability of a given edge ( $E$ ) given the data which is given by,

$$p(E|D) = \sum_G p(E|G)p(G|D)$$

).

**Usage**

```
CalculateEdgeProbabilities(collection)
```

**Arguments**

collection	A collection object where states are DAGs. See CollectUniqueObjects.
------------	--

**Value**

p\_edge An adjacency matrix representing the edge probabilities.

---

CalculateFeatureProbability

*Collect DAG feature probability.*


---

### Description

Calculate the feature ( $f$ ) probability whereby  $p(f|D) = \sum_{G \in \mathcal{G}} p(G|D)p(f|G)$ .

### Usage

```
CalculateFeatureProbability(collection, p_feature)
```

### Arguments

collection	A collection of unique objects. See CollectUniqueObjects.
p_feature	A function that takes an adjacency matrix and collection object and returns a numeric value equal to $p(f G)$ . Therefore, it must be of the form $p\_feature(dag)$ .

### Value

p\_post\_feature A numeric value representing the posterior probability of the feature.

---

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 objects.*

---

**Description**

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

**Usage**

```
CollectUniqueObjects(x)
```

**Arguments**

x	A dagmc_chains or dagmc_chain object.
---	---------------------------------------



## Details

This gets the unique set of states in a dagmc\_object referred to as objects ( $\mathcal{O}$ ). Then estimates the log of the normalisation constant assuming  $\tilde{Z}_{\mathcal{O}} = \sum_s^S p(\mathcal{O}_s)p(D|\mathcal{O}_s)$  where  $\{\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, \dots, \mathcal{O}_S\}$  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  $Z = \sum_{G \in \mathcal{G}} p(G)p(D|G)$  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  $X_i$ .

After the normalisation constant has been estimated we then estimate the log probability of each object 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:

- state: List of unique states.
- log\_evidence\_state: Numeric value representing the evidence calculated from the states.
- log\_state\_score: Vector with the log scores for each state.

---

CreateScorer

*Scorer constructor.*

---

## Description

Scorer constructor.

## Usage

```
CreateScorer(
  scorer = BNLearnScorer,
  ...,
  max_parents = Inf,
  blacklist = NULL,
  whitelist = NULL,
  cache = FALSE,
  nthreads = 1
)
```

**Arguments**

scorer	A scorer function that takes (node, parents) as parameters. Default is BNLearn-Scorer.
...	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.
nthreads	Number of threads used to construct cache.

**Examples**

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

---

DAGtoCPDAG	<i>Convert DAG to CPDAG.</i>
------------	------------------------------

---

**Description**

Convert DAG to CPDAG.

**Usage**

```
DAGtoCPDAG(x)
```

**Arguments**

x	A matrix, dagmc_chain, or dagmc_chains object. When it is a chain(s) object the state must be an adjacency matrix.
---	--

**Value**

x Returns same object type converted to a CPDAG.

---

DefaultProposal	<i>Default proposal constructor.</i>
-----------------	--------------------------------------

---

**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	<i>Find nodes with changed parent combinations between different labelled partitions.</i>
------------------	---

---

**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	<i>Flatten list of chains.</i>
---------------	--------------------------------

---

**Description**

Flatten list of chains.

**Usage**

```
FlattenChains(chains)
```

**Arguments**

chains	MCMC chains.
--------	--------------

---

GetEmptyDAG	<i>Get an empty DAG given a set of nodes.</i>
-------------	---

---

**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).

---

GetIncrementalScoringEdges

*Get the score of the empty DAG*


---

**Description**

Get the score of the empty DAG

**Usage**

```
GetIncrementalScoringEdges(scorer, cutoff = 0)
```

**Arguments**

scorer	A scorer object.
cutoff	A score cutoff.

**Value**

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

---

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). This is rarely used now as we found that it blacklists edges that have significant dependencies but are not in the top  $n$  edges. We prefer the GetIncrementalScoringEdges method.

**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.

---

GetMAP	<i>Get the maximum a posteriori state.</i>
--------	--

---

**Description**

Get the maximum a posteriori state.

**Usage**

GetMAP(x)

**Arguments**

x	A collection of unique objects. See CollectUniqueObjects.
---	---

**Value**

maps A list with the adjacency matrix for the map and it's posterior probability. It is possible for it to return multiple DAGs. The list has elements;

- state: List of MAP DAGs.
- log\_p: Numeric vector with the log posterior probability for each state.

---

GetNodePartition	<i>Get a node's partition element number.</i>
------------------	---

---

**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 lambda in Kuipers & Moffa (2015).

**Usage**

GetOrderedPartition(partitioned\_nodes)

**Arguments**

partitioned\_nodes  
    Labelled partition.

**Value**

Ordered partition.

---

GetParentCombinations	<i>Get parent combinations for a given node.</i>
-----------------------	--

---

**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.

---

GetParentsKey	<i>Get parents key.</i>
---------------	-------------------------

---

**Description**

TODO: The in function is quite slow. May need to make this a for loop in C++.

**Usage**

```
GetParentsKey(parents, nodes)
```

**Arguments**

parents	A character vector of the parent nodes.
nodes	A character vector for all nodes.



---

`GetPartiallyIncrementalEdges`*Get partially incremental scoring edges.*

---

**Description**

Get the positive incremental scoring edges after conditioning on all other variables.

**Usage**

```
GetPartiallyIncrementalEdges(scorer, cutoff = 0)
```

**Arguments**

scorer            A scorer object.

cutoff            A cutoff value for the blacklist. Less than this value is blacklisted.

---

`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	<i>Get nodes in a partition element.</i>
-------------------	--

---

**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	<i>Get nodes that have restricted parents.</i>
--------------------	--

---

**Description**

Get nodes that have restricted parents.

**Usage**

```
GetRestrictedNodes(list)
```

**Arguments**

list	A black or white list.
------	------------------------

---

GetRestrictedParents	<i>Get black or white listed parents.</i>
----------------------	---

---

**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	<i>Log-Sum-Exponential calculation using the trick that limits underflow issues.</i>
-----------	--

---

**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	<i>Node move proposal.</i>
----------	----------------------------

---

**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. One step implementation of the tempered partition MCMC. This acts as a constructor.*

---

### Description

This is a constructor for a single Tempered Partition MCMC step. The function constructs an environment with the proposal, inverse temperature, and verbose flag. It then returns a function which takes the current\_state and a scorer object. This only allows the scores to be raised to a constant temperature for every step.

### Usage

```
PartitionMCMC(
  proposal = NULL,
  temperature = 1,
  prerejection = TRUE,
  verbose = TRUE
)
```

### Arguments

proposal	Proposal function. Default is the DefaultProposal.
temperature	Numeric value representing the temperature to raise the score to.
prerejection	Boolean flag to reject due to the proposal disobeying the black or white lists. Only set to FALSE if you want to understand how often you are proposing states that disobey the black or white lists. Can be useful for debugging or understanding the efficiency of specific proposal distributions.
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 = NodeMove, temperature = 1.0)
pmcmc(current_state, scorer)
```

---

PartitionSplitJoin	<i>Partition split or join constructor.</i>
--------------------	---

---

**Description**

Partition split or join constructor.

**Usage**

```
PartitionSplitJoin(partitioned_nodes)
```

**Arguments**

partitioned\_nodes  
Labelled partition.

---

PartitiontoDAG	<i>Sample DAGs from labelled partitions.</i>
----------------	--

---

**Description**

Sample DAGs from labelled partitions.

**Usage**

```
PartitiontoDAG(partitions, scorer)
```

**Arguments**

partitions      A dagms\_chains, dagmc\_chain, or matrix.  
scorer          A scorer object.

---

PlotScoreTrace	<i>Plot the score trace.</i>
----------------	------------------------------

---

### 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	<i>Analysis of chains. Equilibrium states.</i>
-------------------	--

---

### 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	<i>Propose individual node movement.</i>
-----------------	--

---

### 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	<i>Propose a split or join of two partitions.</i>
---------------------------	---

---

### 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	<i>Propose that two nodes swap partition elements.</i>
-----------------	--

---

**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	<i>Sample a single chain.</i>
-------------	-------------------------------

---

**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.

**Value**

chain A dagmc\_chain object.

**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)
```

---

SampleChains	<i>Sample multiple chains in parallel.</i>
--------------	--

---

**Description**

Sample multiple chains in parallel.

**Usage**

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

**Arguments**

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

**Value**

`chains` A `dagmc_chains` object.

**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 single DAG with from the set of DAGs that is consistent with a labelled partition. That is, given a set of DAGs  $\mathcal{G}$  that is consistent with a labelled partition  $\Lambda$  then it will sample a given DAG  $G$  where  $G \in \mathcal{G}$  with probability  $p(G|D, \Lambda) = p(G)p(D|G) / \sum_{G \in \mathcal{G}} p(G)p(D|G)$ .

**Usage**

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

**Arguments**

```
partitioned_nodes    Labelled partition.
scorer               Scorer object.
```

**Value**

A list with elements:

- state Adjacency matrix.
- log\_score Score of the sampled DAG.

**Examples**

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

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

scorer <- CreateScorer(data = data)

SampleDAGFromLabelledPartition(partitioned_nodes, 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 other-
                   wise.
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	<i>Calculate score tables for (node, parents) combinations.</i>
----------------	---

---

**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	<i>Swap node proposal.</i>
----------	----------------------------

---

**Description**

Swap node proposal.

**Usage**

```
SwapNode(partitioned_nodes)
```

**Arguments**

partitioned\_nodes  
 Labelled partition.

---

toBNLearn	<i>Convert to bnlearn object.</i>
-----------	-----------------------------------

---

**Description**

Convert to bnlearn object.

**Usage**

```
toBNLearn(x)
```

**Arguments**

x  
 Adjacency matrix.

**Value**

bn\_obj A bn object.

**Examples**

```
adj <- UniformlySampleDAG(c('A', 'B', 'C'))
bn_obj <- toBNLearn(adj)
```

---

toMatrix	<i>Convert network to an adjacency matrix.</i>
----------	--

---

**Description**

Convert a bnlearn or igraph object to an adjacency matrix.

**Usage**

```
toMatrix(network)
```

**Arguments**

network            A network object from bnlearn or igraph.

**Value**

An adjacency matrix representation of network.

**Examples**

```
toMatrix(bnlearn::empty.graph(LETTERS[1:6]))
toMatrix(igraph::sample_k_regular(10, 2))
```

---

UniformlySampleDAG	<i>Uniformly sample DAG given a set of nodes.</i>
--------------------	---

---

**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).

---

[.dagmc_chain	<i>Indexing with respect to iterations.</i>
---------------	---

---

**Description**

Indexing with respect to iterations.

**Usage**

```
## S3 method for class 'dagmc_chain'
x = list()[i, ...]
```

**Arguments**

x	A dagmc_chain object.
i	An index.
...	ellipsis for extra indexing parameters.

**Value**

chain A dagmc\_chain.

---

[.dagmc_chains	<i>Index a dagmc_chains object with respect to iterations.</i>
----------------	--

---

**Description**

Index a dagmc\_chains object with respect to iterations.

**Usage**

```
## S3 method for class 'dagmc_chains'
x = list()[i, ...]
```

**Arguments**

x	A dagmc_chain object.
i	An index to get the dagmc_chain iterations.
...	ellipsis for extra indexing parameters.

**Value**

chain A dagmc\_chains object.

---

[[.dagmc_chains	<i>Index a dagmc_chains object.</i>
-----------------	-------------------------------------

---

**Description**

Index a dagmc\_chains object.

**Usage**

```
## S3 method for class 'dagmc_chains'  
x[[i, ...]]
```

**Arguments**

x	A dagmc_chains object.
i	An index to get the dagmc_chain.
...	ellipsis for extra indexing parameters.

**Value**

chain A dagmc\_chains object.

# Index

[\[.dagmc\\_chain, 35](#)  
[\[.dagmc\\_chains, 35](#)  
[\[\[.dagmc\\_chains, 36](#)  
  
[BNLearnScorer, 3](#)  
  
[CachedScorer, 3](#)  
[CalculateAcceptanceRates, 4](#)  
[CalculateEdgeProbabilities, 4](#)  
[CalculateFeatureProbability, 5](#)  
[CalculateNodeMoveNeighbourhood, 5](#)  
[CalculateSplitJoinNeighbourhood, 6](#)  
[CalculateStayStillNeighbourhood, 6](#)  
[CalculateSwapAdjacentNodeNeighbourhood, 7](#)  
[CalculateSwapNodeNeighbourhood, 7](#)  
[CheckBlacklistObeyed, 7](#)  
[CheckWhitelistObeyed, 8](#)  
[CollectUniqueObjects, 8](#)  
[CreateScorer, 9](#)  
  
[DAGtoCPDAG, 10](#)  
[DefaultProposal, 11](#)  
  
[FindChangedNodes, 11](#)  
[FlattenChains, 12](#)  
  
[GetEmptyDAG, 12](#)  
[GetIncrementalScoringEdges, 13](#)  
[GetLowestPairwiseScoringEdges, 13](#)  
[GetMAP, 14](#)  
[GetNodePartition, 14](#)  
[GetNumberOfPartitions, 15](#)  
[GetOrderedPartition, 15](#)  
[GetParentCombinations, 16](#)  
[GetParentsKey, 16](#)  
[GetPartiallyIncrementalEdges, 17](#)  
[GetPartitionedNodesFromAdjacencyMatrix, 17](#)  
[GetPartitionNodes, 18](#)  
[GetRestrictedNodes, 18](#)  
[GetRestrictedParents, 18](#)  
  
[LogSumExp, 19](#)  
  
[NodeMove, 19](#)  
  
[OrderPartitionedNodes, 20](#)  
  
[PartitionMCMC, 20](#)  
[PartitionSplitJoin, 21](#)  
[PartitiontoDAG, 21](#)  
[PlotScoreTrace, 22](#)  
[PostProcessChains, 22](#)  
[ProposeNodeMove, 23](#)  
[ProposePartitionSplitJoin, 23](#)  
[ProposeStayStill, 24](#)  
[ProposeSwapAdjacentNode, 24](#)  
[ProposeSwapNode, 25](#)  
  
[SampleChain, 26](#)  
[SampleChains, 26](#)  
[SampleDAGFromLabelledPartition, 27](#)  
[ScoreDAG, 28](#)  
[ScoreDiff, 29](#)  
[ScoreLabelledPartition, 30](#)  
[ScoreNode, 30](#)  
[ScoreTableNode, 31](#)  
[StayStill, 32](#)  
[SwapAdjacentNode, 32](#)  
[SwapNode, 33](#)  
  
[toBNLearn, 33](#)  
[toMatrix, 34](#)  
  
[UniformlySampleDAG, 34](#)