PANDROIDE Branch and bound method for LIMID Inference

Release 1.0.0

David PINAUD, Emilie BIEGAS

May 09, 2021

CONTENTS:

- genindex
- modindex

File containing a class that allows the encapsulation of an influence diagram and its resolution by a branch and bound method

class bandbLIMID.**BranchAndBoundLIMIDInference**(*ID*, *OrdreDecision*, *verbose=False*)
Class that allows the encapsulation of an influence diagram and its resolution by a branch and bound method

ID
The influence diagram to encapsulate

Type pyAgrum.InfluenceDiagram

OrdreDecision

Type list of decision node ids in the order in which the decisions are taken in the influence diagram

SIS(*decisionNodeID*, *ID*)
Function that returns the Sufficient information set for a given decision node and influence diagram

Parameters

- **decisioNodeID** (*int*) – the id of the decision node
- **ID** (*pyAgrum.InfluenceDiagram*) – the ID on which to base the creation of the graph

Returns **SIS** – The sufficient information set of the given decision node in the given influence diagram

Return type set of ints

addCouche(*index*, *root*, *parents_chanceID*, *pile*)
Function that creates a new branch given a root node and the index of the decision node that is the bottom layer to be

Parameters

- **index** (*int*) – index of the decision node that is going to be the bottom layer in the branch
- **root** (chanceNode) – the node that serves as a root of the subtree
- **parents_chanceID** (*list*) – list of the parents of the decision node, used to build the tree before adding the decisions nodes as bottom layer
- **pile** (*list*) – the list of decision nodes still to be processed

Returns the list of nodes that constitutes the new branch

Return type list

branchAndBound()
Function that executes the branch and bound algorithm. It generated the and or graph on the fly and calculates the MEU for every decision nodes/chance nodes for every instanciation possible

checkNameTaken(*bn*, *name*)
Utility function for viewAndOrGraphNoCuts, checks if a name is taken in the BN (we need it because the and or tree has a lot of nodes with the same name)

Parameters

- **bn** (*pyAgrum.BayesNet*) – the bn to check

- **name** (*str*) – the name to check

Returns true if the name is in the bn

Return type bool

createCoucheChance(*parents*, *root*, *contexte*)
Function that builds a new branch by adding recursively layers of chance nodes (only, decision nodes are added with the createCoucheDecision function)

Parameters

- **parents** (*list*) – the set of parents of the decision node for which we are branching (not the one from where we branch but the ones that will be created)
- **root** (chanceNode) – the chance node that is the root of the subtree
- **contexte** (*dict*) – the instanciation path of the root

createCoucheDecision(*root*, *decisionNodeID*, *contexte*, *pile*, *couche*)
Function that builds the layer of decision node at the base of a new branch

Parameters

- **root** (chanceNode) – the root of the subtree
- **decisionNodeID** (*int*) – the id of the decision node (in the ID) for which we must build the layer
- **contexte** (*dict*) – the instanciation path of a decision node
- **pile** (*list*) – the list of decision nodes to expand during the branch and bound, new decisions nodes are added as they are created
- **couche** (*list*) – the branch for which we must create the layer

Returns the branch but now with a layer of decision nodes

Return type list

createRelaxation()
Function that creates the relaxation of the encapsulated influence diagram by adding informations to the ID through the SIS and removing non-required arcs

Returns The relaxed influence diagram

Return type pyAgrum.InfluenceDiagram

createRest(*bn*, *decisionNodeCreated*, *i*)
Function that creates the rest of the BN for viewAndOrGraphNoCuts

Parameters

- **bn** (*pyAgrum.BayesNet*) – the BN
- **decisionNodeCreated** (*list*) – the decision created in a layer
- **i** (*int*) – the index of the current decision node that is being developped

evaluate(*ID*, *evidence*)
Function that makes the inference over an ID given evidence

Parameters

- **ID** (*pyAgrum.InfluenceDiagram*) – The influence diagram to evaluate
- **evidence** (*set of {key:nodeID,value:probability} (cpt)*) – the evidence to set in the inference

Returns The inference object with makeInference() already called

Return type pyAgrum.ShaferShenoyLIMIDInference

findCoucheDeNoeudDeDecision(*decisionNode*, *couches*)
Utility function that allows to find the branch of a certain decision node

Parameters

- **decisionNode** (decisionNode) – the decision node for which we want to find the branch
- **couches** (*list*) – the list of all the branches in the and/or tree

Returns the branch where the decision node is

Return type list or None

findLigneAuDessus(*ligne*)
function that allows to find the layer above the layer "ligne" in a branch

Parameters **ligne** (*list*) – the layer of nodes (chance of decision nodes) for which we wish to find the layer above

Returns the layer of nodes above

Return type list

fromIDToMoralizedAncestral(*decisionNodeID*, *ID*)
Function that, given an influence diagram and a decision node id, creates the corresponding moralized ancestral undirected graph and adds a source and a well node. It is used to generate a graph on which the SIS algorithm can work to return the SIS of the decision node given.

Parameters

- **decisioNodeID** (*int*) – the id of the decision node
- **ID** (*pyAgrum.InfluenceDiagram*) – the ID on which to base the creation of the graph

Returns

- **MoralizedAncestral** (*pyAgrum.UndiGraph*) – the moralized ancestral undirected graph generated
- **alphaXid,BetaYid** (*int*) – the source and well added to the graph

getBNFromID(*idiag: pyAgrum.pyAgrum.InfluenceDiagram*)
Function that gives us the bayesian network for finding posterior probability when doing the backwards inductio, part of the branch and bound

Parameters **idiag** (*pyAgrum.InfluenceDiagram*) – the ID for which we want the bayesian network

Returns the bayesian network generated

Return type pyAgrum.BayesNet()

getDecisionOpt(*decisionNode*)
Function that, given a decisionNode from the And/Or Graph, returns the optimal decision and its MEU value. It is ONLY used internally and is part of the branch and bound algorithm. It is used only on decision nodes that are leaf nodes in the And/Or Graph that are already evaluated. Do NOT use this function to get the optimal decision of a decision Node.

Parameters **decisionNode** (andOrGraph.decisionNode) – the decision node, part of the And/Or Graph

Returns

- **decisionOpt** (*Object*) – The optimum decision
- **valeurDecisionOptimale** (*float*) – the value of the optimal decision

getDomain (*NodeID*)
Function that returns the domain in which this node can instanciate

Parameters **NodeID** (*int*) – the id of the node

Returns Domain of the node

Return type list

getNameFromID (*idNode*)
Function that returns the name of a node in the ID from its id

Parameters **idNode** (*int*) – the id of the node

Returns the name of the node

Return type str

getNamesFromID (*listId*)
Function that returns the name of nodes in the influence diagram given their ids

Parameters **listId** (*list of int*) – list of the ids of the node

Returns list of the names of the nodes

Return type list of str

getParents_chanceID (*decisionNodeID*, *nodeADevID*)
Function that returns the parents of a decision node (that is the leaf the branch we want to create)

Parameters

- **decisionNodeID** (*int*) – the id of the decision node in the and or graph
- **nodeADevID** (*int*) – the id of the decision node in the ID

Returns list of parents of the decision node

Return type list

getSIS (*decisioNodeID*)
Function that returns the Sufficient Information Set of a decision node

Parameters **decisioNodeID** (*int*) – the id of the decision node

Returns the Sufficient Information Set of the decision Node

Return type set of ints

induction (*ligne*)
Function that allows to go up through a layer in a branch given a layer ligne. It calculates the values of the chance nodes when the layer consist of chance nodes and the MEU of decision nodes when it consist decision node

Parameters **ligne** (*list*) – layer in which we want to start the going up process

Returns the layer above the layer ligne

Return type list

inductionArriere(*couche*, *pile*, *couches*, *indexPile*)
Function that is called when the algorithm arrives at decision nodes that are leafs in the and/or graph, it allows to go back up through the branch while computing the values of the chance nodes and the MEU of the decision nodes It also prunes branches that upper bound are smaller than the best evaluation. It is a recursive function that uses the induction function.

Parameters

- **couche** (*list*) – the branch we wish to go up through
- **pile** (*list*) – the list of decisions nodes to expand
- **couches** (*list*) – the list of branches present in the and/or tree
- **indexPile** (*int*) – the next decision node to expand

Returns if the algorithm reaches the root of the and or graph, it returns None and the algoritm has finished, otherwise, it returns an int that is the index of next decision node to expand in the pile

Return type int or None

isAllDecisionNodeProcessed(*couches*)
Function that checks if all the decision nodes are processed

Parameters **couches** (*list of list*) – list of all the branches in the and or graph

Returns true if all the decision nodes are processed, false otherwise

Return type bool

setVerbose(*verbose: bool*) → None
Function that allows to set the verbose parameter, true will print the trace, false will not

Parameters **verbose** (*bool*) – the parameter

viewAndOrGraph()
Creates a BN that allows to visualize the and or graph (without branches that have been cut)

Returns the BN

Return type BayesNet

viewAndOrGraphNoCuts()
Creates a BN that allows to visualize the complete and or graph (with branches that have been cut).

Returns the BN

Return type BayesNet

viewCreateCoucheChance(*bn*, *s*, *parents*, *idNodeIDParent*, *idNodeBNParent*)
creates a layer of chance node for viewAndOrGraphNoCuts

Parameters

- **bn** (*pyAgrum.BayesNet*) – the bn in which to add the nodes
- **s** (*str*) – utility string to modulate the name of the nodes (prevent reuse)
- **parents** (*list*) – list of chance nodes that are parents of the decision node
- **idNodeIDParent** (*int*) – id in the influence diagram of the parent of the chance node that is being developped
- **idNodeBNParent** (*int*) – id in the bayesian network of the parent of the chance node that is being developped

viewCreateDecisionCouche(*bn*, *decisionNode*, *root*)
creates a layer of decision node for viewAndOrGraphNoCuts

Parameters

- **bn** (*pyAgrum.BayesNet*) – the bn in which to add the nodes
- **decisionNode** (*int*) – the id of the decision node to add in the layer
- **root** (*int*) – the id of the root of the layer (a chance node)

Returns list of ids of the decision node in the bn

Return type list

class andOrGraph.**andOrGraph**(*ID*, *root*)
Class that emulates an And/Or Graph (in reality its a tree) .. attribute:: ID

The influence diagram we use for the And/Or Graph

type pyAgrum.InfluenceDiagram

root
the root of the And/Or Graph

Type *andOrGraph.chanceNode*

noeuds
The list of nodes in the graph

Type list of andOrGraph.chanceNode and andOrGraph.decisionNode

noeudsChance
The list of andOrGraph.chanceNode in the graph

Type list of andOrGraph.chanceNode

noeudsDecision
The list of andOrGraph.decisionNode in the graph

Type list of andOrGraph.decisionNode

IDNoeudDecisionAndOr
Integer that serves to give ids to the decision nodes (different to their influence ids)

Type int

addNoeudChance(*noeud*)
Function that adds a andOrGraph.chanceNode to the And/Or Graph

Parameters **noeud** (*andOrGraph.chanceNode*) – the chance node to add

addNoeudDecision(*noeud*)
Function that adds a andOrGraph.decisionNode to the And/Or Graph It also sets its And/Or Graph id

Parameters **noeud** (*andOrGraph.decisionNode*) – the decision node to add

getID()
Getter function for the ID attribute

Returns The influence diagram we use for the And/Or Graph

Return type pyAgrum.InfluenceDiagram

getIDNoeudAndOr()
Getter function for the IDNoeudAndOr attribute

Returns Integer that serves to give ids to the decision nodes (different to their influence ids)

Return type int

getNoeud()
Getter function that returns all the And/Or Graph nodes

Returns The list of nodes in the graph

Return type list of andOrGraph.chanceNode and andOrGraph.decisionNode

getNoeudChance()
Getter function for the noeudsChance attribute

Returns The list of andOrGraph.chanceNode in the graph

Return type list of andOrGraph.chanceNode

getNoeudDecision()
Getter function for the ID attribute

Returns The influence diagram we use for the And/Or Graph

Return type pyAgrum.InfluenceDiagram

getNoeudDecisionAndOrIDs()
Getter function that returns all the And/Or Graph ids

Returns The ids of the nodes in the And/Or Graph

Return type list of ints

getNoeudWithIdAndOr(*id_andOr*)
Getter function for that returns a decision node given its And/Or Graph id (not the Influence Diagram one)

Returns the decision node corresponding to the given id

Return type *andOrGraph.decisionNode*

getRoot()
Getter function for the root attribute

Returns the root of the And/Or Graph

Return type *andOrGraph.chanceNode*

setRoot(*root*)
Setter function for the root attribute

Parameters **root** (andOrGraph.chanceNode) – the root of the And/Or Graph

class andOrGraph.**chanceNode**(*Id*, *support*, *parent*, *valeurParent*, *contexte*, *id_andOr*)
AND node for the And/Or Graph .. attribute:: Id

the id of this node in the corresponding influence Diagram

type int

support
the domain of this node

Type list

valeur
the value of this node calculated during the induction process

Type float

parent
the parent of this node in the andOrGraph

Type *chanceNode* or *decisionNode*

probabilitiesPosteriori
The posterior probabilities calculated during the induction process ; key=a domainValue of this node value: a float

Type dict

contexte
The instanciation context of this node ; key = id of a node in the influence diagram, value = the instanciation value of said node

Type dict

id_andOr
the id of this node in the andOrGraph

Type int

childs
children of this node ; key=domainValue, value=chanceNode or decisionNode

Type dict

getChilds()
childs of the node in the And/Or Graph :returns: key=domainValue, value=chanceNode or decisionNode :rtype: dict

getContexte()
The instanciation context of this node

Returns key = id of a node in the influence diagram, value = the instanciation value of said node

Return type dict

getId_andOr()

Returns the id of the node in the AND/OR Graph

Return type int

getNodeID()

Returns the id of the node in the influence diagram

Return type int

getParent()
parent of the node

Returns the parent of the node

Return type *chanceNode* or *decisionNode*

getProbabilitiesPosteriori()
The posterior probabilities calculated during the induction process

Returns key=a domainValue of this node value: a float

Return type dict

getSupport()

Returns domain of the node

Return type list

getValeur()
the value of the chance node calculated in the induction process

Returns value of the chance node

Return type float

class andOrGraph.**decisionNode**(*Id*, *contexte*, *parent*, *support*, *id_andOr*)
OR node for the And/Or Graph .. attribute:: Id

the id of this node in the corresponding influence Diagram

type int

support
the domain of this node

Type list

decisionOptimale
the optimal decision of this node

Type any

ValeurDecisionOptimale
the value of the optimal decision of this node calculated during the induction process

Type float

parent
the parent of this node in the andOrGraph

Type *chanceNode* or *decisionNode*

contexte
The instanciation context of this node ; key = id of a node in the influence diagram, value = the instanciation value of said node

Type dict

id_andOr
the id of this node in the andOrGraph

Type int

doNotDevelop
list of domain values that should not be developped (its upper bound is smaller than the evaluation of another branch corresponding to a domain value of this node)

Type list

inference
the inference object on which we used to calculated the value of the node (only if this node is a leaf)

Type pyAgrum.ShaferShenoyLIMIDInference

enfants
children of this node ; key=domainValue, value=chanceNode or decisionNode

Type dict

evaluation
evaluation of this node ; key=domainValue, value=(mean,variance)

Type dict

borneSup
upper bounds for the branches of this node (one for each domain value and only if this node is a leaf), key=domainValue, value=(mean,variance) ;

Type dict

getBorneSup()
Getter function for the upper valuation of the node (only is not a leaf)

Returns key=domainValue, value=(mean,variance)

Return type dict

getContexte()
The instanciation context of this node

Returns key = id of a node in the influence diagram, value = the instanciation value of said node

Return type dict

getDecisionOptimale()
Returns the optimal decision for this node

Returns the optimal decision of this node, a value of its domain

Return type any

getEnfants()

Returns key=domainValue, value=chanceNode or decisionNode

Return type dict

getEvaluation()
Getter function for the valuation of the node

Returns key=domainValue, value=(mean,variance)

Return type dict

getId_andOr()

Returns the id of the node in the AND/OR Graph

Return type int

getInference()
returns the Shafer Shenoy Object used to make the inference on this node (only if it is a leaf node)

Returns the inference object

Return type pyAgrum.ShaferShenoyLIMIDInference

getNodeID()

Returns the id of the node in the influence diagram

Return type int

getParent()
parent of the node

Returns the parent of the node

Return type *chanceNode*

getSupport()

Returns domain of the node

Return type list

getValeurDecisionOptimale()
return the value of the optimal decision

Returns the value

Return type float

PYTHON MODULE INDEX

INDEX

I

M

N

O

P

R

S

V