Proceedings of the Twenty-Eighth International Florida Artificial Intelligence Research Society Conference

On-Line Learning of Multi-Valued Decision Diagram

Jean-Christophe Magnan and **Pierre-Henri Wuillemin**
Laboratoire d'Informatique de Paris VI – UPMC – France

Abstract

In the domain of decision theoretic planning, the factored framework (Factored Markov Decision Process, FMDP) has produced optimized algorithms using Decision Trees (Structured Value Iteration (SVI), Structured Policy Iteration (SPI)) or Algebraic Decision Diagrams (Stochastic Planning Using Decision Diagrams (SPUDD)). Since it may be difficult to determine the factored models of such complex stochastic environments, the framework SDYNA, which combines planning and learning algorithms using structured representations was proposed. Yet, the state-of-the-art algorithms for incremental learning, for structured decision theoretic planning or for reinforcement learning require the problem to be specified only with binary variables and/or use data structures that can be improved in term of compactness. Recently, Multi-Valued Decision Diagrams (MDDs) were proposed as more efficient data structures and planning algorithms dedicated to these data structures were provided. The purpose of this article is to study how to incrementally learn such compact factored models on discrete domains. Its main result is an online learning algorithm for MDDs that shows significant improvements both in term of quality of the learned model and in time. Finally, we show that our algorithm leads to a SDYNA framework for simultaneous learning and planning using MDDs.

Introduction

In decision-theoretic planning, the Markov Decision Process (MDP) is a widely used framework that formalizes the interactions of an agent with a stochastic environment. A MDP is commonly used to find an optimal policy, i.e. the best action for the agent to do in each configuration of the environment (state). Two algorithms named Value Iteration (VI (Bellman 1957)) and Policy Iteration (PI (Howard 1960)) exploit such models to find optimal policies. Each step of these algorithms has a linear time complexity in the size of the state space. However, for realistic problems, the size of the state space tends to be very large. State spaces are indeed often multi-dimensional and then grow exponentially as the number of variables (dimensions) characterizing these problems increases. VI and PI inevitably fall under the famous Bellman's "Curse of Dimensionality" (Bellman 1957). It becomes unfeasible to find the optimal solution for realistic problems.

Many relevant solutions have emerged to handle this growth: for instance by formalizing abstractions in the state space. Through these abstractions, large sets of states in which the agent mostly behaves the same are aggregated. As a result, the number of states to visit during an iteration of VI or PI drastically diminished. Working on this idea, Factored MDPs have proven to be efficient. Structured VI (SVI) and Structured PI (SPI) (Boutilier, Dean, and Hanks 1999) proposed to employ Decision Trees (DTs) as a base for the abstraction process. More recently, Stochastic Planning using Decision Diagrams (SPUDD, (Hoey et al. 1999)) and Multi-valued Decision Diagrams (SPUMDD, (Magnan and Wuillemin 2013)) achieved better results using Algebraic Decision Diagrams (ADDs) or Multi-Valued Decision Diagrams (MDDs).

These algorithms involve to know as a prior the factored model of the problem to be solved. In the Reinforcement Learning framework, the agent has the charge to discover the world and to learn by itself the optimal policy. Model-based approaches such as DYNA and DYNA-Q (Sutton 1990) propose that the agent iteratively learns a description of the world as a Markov Decision Process and then apply either VI or PI to come up with an optimal policy. Of course, these algorithms face difficulties when scaling up and it becomes necessary to provide the ability to handle factored representations ((Strehl, Diuk, and Littman 2007; Chakraborty and Stone 2011)). With SPITI, Degris, Sigaud, and Wuillemin(2006a) extends the framework to factored representations using DTs. An incremental learning algorithm ITI (Utgoff, Berkman, and Clouse 1997) is used to learn the DTs defining the MDPs as the agent experiments the world. DTs are however less efficient in terms of size and, as a consequence, of handling than ADDs or MDDs (see Figure 1). Hoey et al.(1999) and Magnan and Wuillemin(2013) provide planning algorithms using these data structures.

Learning such models is difficult, particularly for systems with a huge number of states. It may even be impossible to build the complete database needed to elaborate these models. Generally speaking, there are many cases where an online (or incremental) learning process is more appropriate. To the best of our knowledge there is no known algorithm for incremental learning of Multi-Valued Decisions Diagrams. The main contribution of this paper is an algorithm addressing this issue. This article is organized as follows: Section

2 covers the frameworks for incremental learning of compact and factored models. Section 3 describes IMDDI, our proposition to incrementally learn MDDs. Finally, the experimentations in Section 4 illustrate the efficiency of IMDDI both in terms of quality of the learned models and in terms of duration of a learning step.

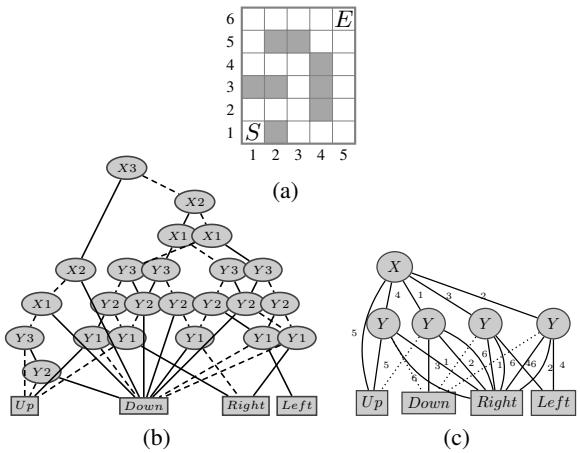


Figure 1: Same optimal Policy for going from S to E in the maze (a) represented by (b) a Arithmetic Decision Diagram (ADD) and (c) by a Multi-Valued Decision Diagram (MDD). The compactness as much as the readability of the solution are clearly improved with MDDs.

Learning Factored Models while planning

A Factored Markov Decision Process (FMDP, (Puterman 2005)) is a discrete time control process that models a stochastic system in which an agent sequentially performs actions. A set of variables $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$ is used to unequivocally characterize the states of this system. Performing an action a in any state $\mathbf{x}$ triggers a transition from $\mathbf{x}$ to a new state $\mathbf{x}'$. The transition probability $P(\mathbf{x}'|a, \mathbf{x})$ describes this evolution. The reward function $r(\mathbf{x}', a, \mathbf{x})$ evaluates the relevance of this transition. The objective is to find a policy that maximizes the total expected reward. Boutilier, Dean, and Hanks(1999) and Hoey et al.(1999) show that factored representations of the transition probabilities and reward function using Decision Trees (DTs) or Algebraic Decision Diagrams (ADDs) are effective. They provide modified resolution algorithms which increase the size of solvable problems (see Figure 1).

Planning while Learning (SDYNA)

The representation of the world (reward function and transition probabilities) may not be known because of a lack of data (the state space has not been explored yet) or because of the nature of the data (typically for online process such as stock market prediction). This remark leads to the implementation of online algorithms that have to take into account each new observation ξ. The reinforcement learning framework has proposed to learn these functions by trial-and-error during the experiment (Dyna, Dyna-Q (Sutton 1990)). In (Degris, Sigaud, and Wuillemin 2006b), the general architecture SDYNA integrates planning, acting and learning using factored representations as described in Algorithm 1.

Algorithm 1: SDYNA global architecture

```
foreach time step t do
    s ← current state;
    a ← π_t(s);   // π_t is the current strategy
    Perform a, observe s' and r and define ξ = (s, a, s', r);
    Incremental "factored" learning from observation ξ;
    "Factored" planning new π_{t+1};
```

In order to implement a SDYNA architecture, one has to choose a structured representation of the problem (DT, ADDs, etc.) and to provide an optimal policy search algorithm and an incremental learning algorithm for the data structure. The following subsections will cover learning algorithm for factored models.

Incremental Tree Induction (ITI)

There are many algorithms to learn a tree from a set of observations : CART (Breiman et al. 1984), C4.5 (Quinlan 1993), etc. These approaches require to have the complete set prior to the learning. They are not able to provide adaptation of the tree to any new observation. A windowed approach could be implemented but it would require to rebuild the tree from scratch for each new observation. ITI is an algorithm that fulfill this need of online adaptation to new observations.

In ITI, each node N of the learned DT contains a set of observations Ω_N that are compatible with the instantiation in N. For instance, the sets of observations installed in the leaves of a tree form a partition of the set of all the observations Ω. Then adding a new observation ξ means to update any set compatible with ξ and then to use it to update the structure of the tree if needed. The structure depends itself on these sets of observations : an internal node N contains the "best" (not yet instantiated) variable that separates the set of observations Ω_N. To select the variable, ITI uses the Information Gain Ratio as a criterion and compare the different distributions created by installing each free variable in the node. We refer the readers to (Utgoff, Berkman, and Clouse 1997) for a much more complete presentation of this algorithm.

Learning MDDs

Several articles address the problem of learning Multi-Valued Decision Diagrams. Oliver(1993) proposes to firstly build a tree using the state-of-the-art algorithms (CART, C4.5). Then this tree is transformed in order to facilitate the search for isomorphic subtrees. Eventually, these subtrees are merged using the Minimum Description Lenght principle as a criterion. Kohavi and Li(1995) directly builds an ordered tree and then reduced it using its own set of rules.

These algorithms do not cope with the issue of online learning since the trees are learned from fixed databases. Therefore, adding new observations to the database demands to learn a new whole tree. Hence these approaches can not be good candidates for a SDYNA architecture. Being able to review the tree, and if and only if necessary its reduced version the MDD, without having to go through the whole database

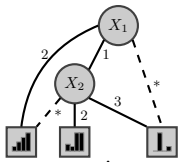


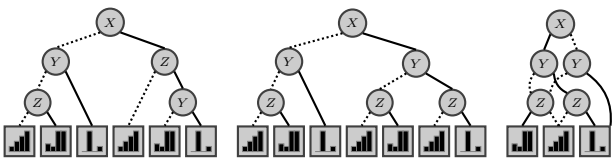
Figure 2: An MDD representing a probability distribution $P(Y|X_1, X_2)$. The leaves contain probability distributions for Y. This MDD states that $P(Y|X_1 = 2) = P(Y|X_1 = 1, X_2 \notin \{2, 3\})$ and that $P(Y|X_1 \notin \{1, 2\}) = P(Y|X_1 = 1, X_2 = 3)$. Those equalities represent Context-Specific Independence in P (Boutilier et al. 1996).

would be a great asset that has not been proposed yet to the best of our knowledge. The next section covers our results on that matter.

On-line Learning of Multi-valued Decision Diagrams

In this section we describe IMDDI, a novel algorithm for the incremental learning of MDDs. Without any loss of generality, this presentation will focus on the estimation of the probability distribution of a multi-valued variable Y according to a set of multi-valued variables $\mathbf{X} = \{X_1, \ldots, X_n\}$ (see Figure 2). Indeed, learning the transition model of a factored MDP consists in learning several conditional probability distributions (Boutilier, Dean, and Hanks 1999). Moreover, learning other functions such as the reward function is done using very similar algorithms.

There are two major differences between a DT and an MDD : first, an MDD is structured by a global order on the variables. Variables must appear on each branch of the MDD w.r.t this global order. This order has a large impact on the compactness of the MDD. We call Ordered Decision Tree (ODT) a DT with this constraint of a global order on the variables. Second, an MDD merges sub-trees together in order to be reduced. The complexity of reducing an ODT T into an MDD is $O(|T| \cdot \log |T|)$ where $|T|$ is the number of nodes in T (Bryant 1986). Figure 3 depicts the transformation from a DT into an MDD via an ODT. A first incremental algorithm to learn MDD (named ITI+DD later) could be i) to simply use ITI to learn the DT, ii) then to choose an order and to build the ODT and finally iii) to build the MDD at each step as it has been proposed for non incremental learning of MDDs by (Oliver 1993). However the search for an optimum global order is a NP-hard algorithm. Moreover, it is possible to just update an estimation of an efficient global order. This is a key point in our algorithm IMDDI : we propose a modified version of ITI that handles an ordered decision tree instead of a tree. Then, in a second part, whenever needed, the MDD will be build from this ODT.



(a) (b) (c)

Figure 3: A probability distribution represented as (a) a DT, (b) an ODT ($X \succ Y \succ Z$) and (c) an MDD (with the same order).

Incremental Induction of an Ordered Tree

IMDDI is based on a version of ITI but the main features have been revised and are described in the following subsections: how to select the variable that will be installed in a node; how to incrementally integrate a new observation ξ and how to update the structure w.r.t. this new observation.

Variable Selection As selection criteria, both G-statistic (Mingers 1989) and χ^2 statistic have no bias toward multi-valued variable (White and Liu 1994). G and χ^2 tests are close when the size of the sample is big enough but (Dunning 1993) argues that G-statistic is superior to the χ^2 statistic for dealing with rare events.

Let N be the node on which we want to either install a variable (if N is a leaf) or ensure that the current variable is the most pertinent. Let Ω_N be the set of associated observations on which we rely to perform our selection (see below Algorithm 2 to understand how these databases are extracted). Let $\mathbf{V}_N$ be the set of variables that could be installed on node N. For each variable $X_i \in \mathbf{V}_N$, N keeps a contingency table giving the samples size $n_{x_i,y}$ for each combination of X_i and Y values. The G-statistic is then computed in the following way :

$$G(X_i) = 2 \cdot \sum_{x_i \in DomX_i} \sum_{y \in DomY} n_{x_i,y} \ln \frac{n_{x_i,y} \cdot |\Omega_N|}{n_{.,y} n_{x_i,.}} \quad (1)$$

where $n_{.,y} = \sum_{x_i} n_{x_i,y}$ and $n_{x_i,.} = \sum_y n_{x_i,y}$.

To decide among every X_i which one should be install as the test, IMDDI compares the p-values associated to these computed G-statistics: variables with a high number of values tend to have a high G-statistic whereas variables with a low number of G-statistic has a low score. The uses of p-values avoids this bias towards multi-valued variables since the p-values "normalizes" every G-statistics by integrating degrees of freedom. Furthermore, as the χ^2, the G-statistic has an interesting feature : it can also be used as a pre-pruning criterion in order to prevent the tree from growing unreasonably. To sum up, the variable with the highest p-value will be selected. If this p-value is higher than a fixed threshold, we install that variable as a test for the node. If it is lower than the threshold and node N is not a leaf, N is turned into a leaf.

Adding a new observation ξ An observation ξ is an instantiation of all the variables of interest $\langle X_1, \cdots, X_n, Y \rangle$. By construction, there exists a unique path form the root to a leaf of the ODT that represents partial instantiations compatible with ξ. Adding ξ to the ODT consists in updating Ω_N and the G-statistics of every node N of that path. With X_N the variable installed in an internal node N, $p_G^N(X_i)$ the p-value for a variable $X_i \in \mathbf{V}_N$ in the node N, $c_N(v)$ the child of node N for the value v of X_N and finally $\xi(A)$ the value for the variable A in ξ, Algorithm 2 describes this update of the internal structure of the ODT.

Updating the ODT Once the statistics have been updated, a last step consists in revising the tree topology. Due to the insertion of the new observation, a revision of the variables

Algorithm 2: AddObs (addition of an observation ξ)

Data: the observation $\xi = \{x_1, \ldots, x_n, y\}$ and the ODT T
Node N $\leftarrow$ root of T;
repeat
  Add ξ to Ω_N;
  foreach *variable* $X_i \in \mathbf{V}_N$ **do**
    Update $p_G^N(X_i)$;
  if *N is not a leaf* **then**
    Node N $\leftarrow c_N(\xi(X_i))$;
until *N is a leaf*;

previously installed may be necessary. However, this revision has to take into account that the order is global. To update the ODT, IMDDI must (i) find a (as good as possible) global order, (ii) ensure that this order is respected on every branch and (iii) ensure that the best possible test is installed on every node. A last requirement is that for any observation that will not change the structure, this operation should be as simple as possible. As an incremental algorithm, IMDDI infers a relevant global order while keeping the possibility to revise it. To fulfill these requirements, the strategy we propose is to check variable by variable the relevance of their position in the current global order. While the variables keep their positions, no structural modifications will be performed in the ODT.

To decide which variable should be the next one in the global order, each remaining variable has to be scored using G-statistics on each node but later aggregated on many nodes within a boundary $\mathcal{B}$. $\mathcal{B}$ is the set of nodes where a variable must be installed w.r.t. the updated global order at each iteration. The boundary is initialized as the root node (singleton) and will contain the leaves of the ODT at the end of this step. IMDDI computes an aggregated score by summing up the p-value computed on each node $N \in \mathcal{B}$ weighted by the proportion of observations on that node ($|\Omega_N|$) compared to the total number of observation added to the tree ($|\Omega|$). The variable with the highest score is then chosen and becomes the next variable in the updated global order. Then, for every node of the boundary, this chosen variable will be installed if the p-value is above a fixed threshold. This installation is done the same way it is done in ITI (see (Utgoff, Berkman, and Clouse 1997) for further details). If the variable is effectively installed, the node is removed from the boundary, replaced by all its children. Once this iteration is over, nodes in the boundary will not be able to choose to install this variable later.

The stopping criterion for the Algorithm 3 has to take into account two cases : either all variables of $\mathbf{X}$ have been added to the updated global order or no variables can be installed on any node of the current boundary (i.e. all the p-values are below the threshold). When it stops, the boundary contains all the leaves of the updated ODT. If the new observation does not change the structure of the ODT, the only computations performed by the Algorithm 3 are weighted sums on each boundary from the root to the leaves of the tree.

From the ODT to the MDD

Once the ODT is generated, the next step is to reduced it into a MDD. Algorithm 4) merges the isomorphic subtrees with a

Algorithm 3: UpdateODT (updating the structure)

Data: an ODT T after adding ξ (with Algorithm 2)
$\mathcal{B} = \{$ root R of $T\}$; `// boundary`
$\mathcal{F} = \mathbf{X}$; `// Set of variables`
repeat
  foreach *variable* $X_i \in \mathcal{F}$ **do**
    $p_G(X_i) = \sum_{N \in \mathcal{B}} \frac{|\Omega_N|}{|\Omega|} \cdot p_G^N(X_i)$;
  $V \leftarrow \arg\min_{X_i \in \mathcal{F}} p_G(X_i)$;
  $\mathcal{B}' \leftarrow \mathcal{B}$;
  foreach $N \in \mathcal{B}$ **do**
    if $p_G^N(V) \geq \tau_1$ **then**
      Install V in node N;
      $\mathcal{B}' \leftarrow \mathcal{B}' \setminus \{N\} \cup \bigcup_{v \in Dom(V)} c_N(v)$
  $\mathcal{B} \leftarrow \mathcal{B}'$;
  $\mathcal{F} \leftarrow \mathcal{F} \setminus \{V\}$;
until $\mathcal{F} = \emptyset$ *or no variable in* $\mathcal{F}$ *can be installed in* $\mathcal{B}$;

bottom-up polynomial (in time) algorithm which starts with the merging of every leaves with similar probability distributions. Then, for every variable X_i, going backward in the global order, two nodes N and N' bound to X_i are merged if they have the same children. At last, if a node has only one child then it is redundant and is replaced by arcs outgoing from its parents to its unique child.

Algorithm 4: Reduce (merge isomorphic subtrees)

Data: an ODT T
repeat
  $(U^*, V^*) = \arg\min_{(U,V)} \text{leaves}(\max(p_U^G, p_V^G))$;
  if $\max(p_G^{U^*}, p_G^{V^*}) \leq \tau_2$ **then**
    Merge U^* and V^*;
until $\nexists$ *two leaves that can merge*;
foreach $X_i \in \mathbf{X}$ *backward w.r.t the order* **do**
  foreach N_{X_i}, N'_{X_i} *with* X_i *as installed variable* **do**
    if $\forall x_i \in Dom(X_i), c_{N_{X_i}}(x_i) = c_{N'_{X_i}}(x_i)$ **then**
      N_{X_i} and N'_{X_i} are merged ;
  foreach N_{X_i} *with* X_i *as installed variable* **do**
    if $\forall x_i^k, x_i^l \in Dom(X_i)$, $c_{N_{X_i}}(x_i^k) = c_{N_{X_i}}(x_i^l)$ **then**
      Replace N_{X_i} by arcs outgoing from the parents to the unique child.

The first stage which consists in merging similar leaves together has to be addressed more specifically. If the leaves of the MDD were discrete values, merging these leaves would be very simple. However, in our framework, each leaf is a probability distribution over the variable Y. As a consequence, we need a test to decide whether or not two probability distributions are similar.

Let U and V be the two leaves we want to merge, let $n_{U,y}$ be the samples size for the value $y \in Dom(Y)$ and $N_U = |\Omega_U|$ be the total number of observations on the leaf U. Merging U and V would produce a new node W. It follows that $\forall y \in Dom(Y)$, $n_{W,y} = n_{U,y} + n_{V,y}$ and $N_W = N_U + N_V$. To determine whether or not we should merge U and V into W, we compute for both leaves a G-statistic : $\forall L \in \{U, V\}, G_L = 2 \cdot \sum_y n_{L,y} \ln \frac{n_{L,y}}{e_{L,y}}$ where $e_{L,y} = n_{W,y} \frac{N_L}{N_W}$. Note that we have to scale down the quantity $n_{W,y}$ to be compared to the quantity $n_{L,y}$. We propose a

greedy algorithm on the leaves of the ODT (Algorithm 4): a couple of p-values (p_G^U, p_G^V) is computed for every possible couple of leaves (U, V). Then the best candidate for merging is $(U^*, V^*) = \arg\min_{(U,V)} \max\left(p_G^U, p_G^V\right)$. This criterion selects the couple with the lowest high dissimilarity in the probability distributions. If both $p_G^{U^*}$ an $p_G^{V^*}$ are smaller than the threshold then the nodes are merged and the process is repeated. The stopping criterion is the absence of merging during an iteration.

Algorithm 5: Incremental MDD Induction (IMDDI) for $P(Y|X_1, \ldots, X_n)$

Data: a data stream of observations : $\xi = (X_1, \ldots, X_n, Y)$
foreach $\xi =$ *next observation* **do**
  AddObs(ξ); `// see Algorithm 2`
  UpdateODT(); `// see Algorithm 3`
  if *change is needed* **then**
    Reduce(); `// see Algorithm 4`

Algorithm 5 presents the complete IMDDI algorithm. Simple complexity considerations show that IMDDI is an algorithm dedicated for online learning: the size of the base of observations Ω has no direct influence on the complexity of the algorithms. Moreover, the more complex algorithm (Algorithm 4) are rarely performed because the ODT will not change often (as described in Algorithm 5) or because the algorithm will just marginally change the structure and then the order will mainly stay the same. The next section will illustrate that the re-evaluation of the MDD does not occur often.

Experiments

In order to analyze the behavior of our algorithm IMDDI, we have compared it to ITI since it is the online learning algorithm for DTs. However, ITI is an algorithm that produces unordered DTs. Hence, we have also compared IMDDI to the extended version ITI+DD which reorders the tree learned by an ITI with the heuristic used in IMDDI and reduces it into an MDD. These three algorithms have been written in Python. Oliver(1993) and Kohavi and Li(1995) propose batch MDD learning algorithms that would also be interesting to compare with. Unfortunately, to the best of our knowledge, no working implementation for these algorithms is accessible.

We have tested our algorithms on databases of observations $\xi = \langle \mathbf{x}, y \rangle$. Each database was associated to a different set of variables $\mathbf{X}$ and to a different distribution $P(Y|\mathbf{X})$. To produce these distributions $P(Y|\mathbf{X})$, we select three different settings for the original model : i) MDD, ii) DT, and iii) Bayesian Networks. In the first setting, IMDDI could find the original model. In the two last settings, the representation of $P(Y|\mathbf{X})$ as a MDD necessary leads to approximation. For the second setting, the challenge is to see if a learned MDD will be able to represent efficiently a DT, despite the absence of explicit isomorphic subpart and global order. Finally a Bayesian Network implies conditional dependencies between Y and the set of variables $\mathbf{X}$ but does not compel the existence of context-specific independences that are existing in MDDs or a DTs.

For all configuration, the domain size of each variable was randomized (up to 5); the size of $\mathbf{X}$ is 10; then the structure and the parameters of each model are randomly chosen. The randomized structure of the model may imply that only a subset of $\mathbf{X}$ is needed for the estimation of Y.

For each setting, 20 random instances were generated. From each instance, databases of 20 000 observations and of 10 000 observations have been generated. The first database is used to learn the model whereas the second one is used to compare the log likelihood of the learned models. The formula being used to compute the log likelihood of a database DB according to a model θ is : $\log P_\theta(DB) = \sum_{\xi \in DB} \log P_\theta(y_\xi | \mathbf{x}_\xi)$.

Quality of the learned models

For a qualitative comparison , we propose to use two criterion : the log likelihood to ensure that IMDDI does not degrade too much the quality of the learn probability distribution in comparison to both ITI and ITI+DD and the size (in term of nodes) of the learned models (MDDs or DTs). As we need a model as compact as possible, this criterion ensure that there is a gain in size of models.

Table 1: Log-likelihood and size comparison (average $\pm$ standard deviation) between IMDDI, ITI and ITI+DD

IMDDI vs ITI	Size	Log Likelihood
MDDs	$63.34\% \pm 9,51\%$	$100.96\% \pm 1,53\%$
DTs	$69.44\% \pm 15.44\%$	$101.93\% \pm 1.89\%$
BNs	$60.51\% \pm 8.73\%$	$100.69\% \pm 1.25\%$

IMDDI vs ITI+DD	Size	Log Likelihood
MDDs	$101.15\% \pm 3,46\%$	$100.01\% \pm 0.04\%$
DTs	$101.52\% \pm 15.22\%$	$100.03\% \pm 0.09\%$
BNs	$105.53\% \pm 5.83\%$	$99.94\% \pm 0.39\%$

Table 1 shows the results obtained by averaging over the twenty instances for each model. The numbers are the relative difference between IMDDI and ITI or ITI+DD. For instance, the first number 63.34% means that in average, IMDDI gives a decrease of 36.66% for the size of the learned model compared to ITI with the first setting (MDD). According to this table, the IMDDI strategy is as interesting as ITI or ITI+DD from this quality of the solution point of view. Moreover, the learned models are clearly more compact than ITI. Compared to ITI+DD, the results both in terms of likelihood and in terms of size do not lead to an improvement for our algorithm. However it has to be reminded that ITI+DD learned the MDD from scratch at each iteration.

Computation time

The other criterion on which we challenged IMDDI and ITI+DD is the time. We compare the time spent to take into account a new observation. Figure 4 shows the average time spent to take into account a new observation when no reduction is applied to the decision diagram. It clearly demonstrates how the time spent to reorder the tree in the ITI+DD

strategy completely renders it inefficient : the reordering part become much slower for ITI+DD as the tree grows.

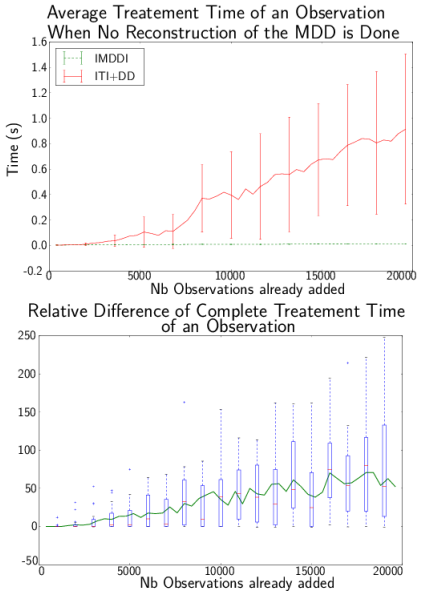


Figure 4: Two comparisons between IMDDI and ITI+DD: (a) times spent to take into account a new observation when no revision of the MDD occurs and (b) evolution of the relative change of the total times to take a new observation into account (curve shows the relative change average).

Steps with reconstruction of the MDDs experimentally represent from 15% to 7% of the observations. In that cases, on all our experiments, the IMDDI step is still shorter than the ITI+DD step. Indeed, in the second part of the Figure 4, we show the relative differences between the total times taken by IMDDI and ITI+DD to handle a new observation : the central curve demonstrates that on average, in spite of the reduction to the MDDs, the IMDDI algorithms remains fifty times faster than the ITI+DD algorithm. These experiments illustrate that our algorithm outperforms ITI in term of quality (same likelihood, better compactness) and outperforms a straightforward ITI+DD in term of computation time.

Conclusion

This article presents IMDDI, the first online learning algorithm for MDDs. As the advantages of MDDs over DTs were demonstrated in Decision Theoretic Planning, such an incremental algorithm became mandatory for their use in the structured Reinforcement Learning framework. The paper details the IMDDI algorithm and its different phases: addition of an observation, update of the structure, and reduction into a decision diagram. It experimentally verifies the compactness and the accuracy of the learned models in comparison with algorithms ITI and ITI+DD. It also illustrates that IMDDI is much faster than a straightforward ITI+DD learning strategy.

It is noteworthy that the IMDDI algorithm needs no prior on the variables: no knowledge about the number of variables or even their domains are required and can be dynamically discovered during the learning.

References

Bellman, R. 1957. *Dynamic Programming*. Princeton, NJ, USA: Princeton University Press, 1 edition.

Boutilier, C.; Friedman, N.; Goldszmidt, M.; and Koller, D. 1996. Context-specific independence in bayesian networks. 115–123.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *JAIR* 11:1–94.

Breiman, L.; Friedman, J.; Olshen, R.; and Stone, C. 1984. *Classification and Regression Trees*. Wadsworth & Brooks.

Bryant, R. E. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers* 35:677–691.

Chakraborty, D., and Stone, P. 2011. Structure learning in ergodic factored mdps without knowledge of the transition function's in-degree. In *ICML-28*, 737–744.

Degris, T.; Sigaud, O.; and Wuillemin, P.-H. 2006a. Chi-square Tests Driven Method for Learning the Structure of Factored MDPs. In *UAI-22*, 122–129. AUAI Press.

Degris, T.; Sigaud, O.; and Wuillemin, P.-H. 2006b. Learning the Structure of Factored Markov Decision Processes in Reinforcement Learning Problems. In *ICML-23*, 257–264.

Dunning, T. 1993. Accurate methods for the statistics of surprise and coincidence. *Comput. Linguist.* 19(1):61–74.

Hoey, J.; St-aubin, R.; Hu, A.; and Boutilier, C. 1999. Spudd: Stochastic planning using decision diagrams. In *UAI-15*, 279–288. Morgan Kaufmann.

Howard, R. A. 1960. *Dynamic Programming and Markov Processes*. MIT Press.

Kohavi, R., and Li, C.-H. 1995. Oblivious decision trees, graphs, and top-down pruning. In *IJCAI-14*, 1071–1077.

Magnan, J.-C., and Wuillemin, P.-H. 2013. Improving Decision Diagrams for Decision Theoretic planning. In *FLAIRS-26*, 621–626.

Mingers, J. 1989. An empirical comparison of selection measures for decision-tree induction. *Machine Learning* 3(4):319–342.

Oliver, J. J. 1993. Decision graphs - an extension of decision trees. In *Proceedings of the Fourth International Workshop on Artificial Intelligence and Statistics*, 343–350.

Puterman, M. 2005. *Markov decision processes: discrete stochastic dynamic programming*. Wiley series in probability and statistics. Wiley-Interscience.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*.

Strehl, E. L.; Diuk, C.; and Littman, M. L. 2007. Efficient structure learning in factored-state mdps. In *AAAI-07*, 645–650.

Sutton, R. S. 1990. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *ICML-7*, 216–224.

Utgoff, P. E.; Berkman, N. C.; and Clouse, J. A. 1997. Decision tree induction based on efficient tree restructuring. *Machine Learning* 29(1):5–44.

White, A. P., and Liu, W. Z. 1994. Technical note: Bias in information-based measures in decision tree induction. *Machine Learning* 15(3):321–329.