Learning the Structure of Factored Markov Decision Processes in Reinforcement Learning Problems

Thomas Degris Thomas.Degris@lip6.fr
Olivier Sigaud Olivier.Sigaud@lip6.fr
Pierre-Henri Wuillemin Pierre-Henri.Wuillemin@lip6.fr
Université Pierre et Marie Curie-Paris6, UMR7606, AnimatLab/LIP6, Paris, F-75005 France ; CNRS, UMR7606, Paris, F-75005 France

Abstract

Recent decision-theoric planning algorithms are able to find optimal solutions in large problems, using Factored Markov Decision Processes (FMDPs). However, these algorithms need a perfect knowledge of the structure of the problem. In this paper, we propose SDYNA, a general framework for addressing large reinforcement learning problems by trial-and-error and with no initial knowledge of their structure. SDYNA integrates incremental planning algorithms based on FMDPs with supervised learning techniques building structured representations of the problem. We describe SPITI, an instantiation of SDYNA, that uses incremental decision tree induction to learn the structure of a problem combined with an incremental version of the *Structured Value Iteration* algorithm. We show that SPITI can build a factored representation of a reinforcement learning problem and may improve the policy faster than tabular reinforcement learning algorithms by exploiting the generalization property of decision tree induction algorithms.

1. Introduction

Markov Decision Processes (MDPs) have been widely used in the Decision-Theoric Planning (DTP) community and are a fundamental framework in the Reinforcement Learning (RL) domain. When the transition and reward functions are known, solution methods based on Dynamic Programming (DP) are effective on small problems. However, they cannot be applied as such to large problems because they require an explicit state space enumeration.

Appearing in *Proceedings of the 23rd International Conference on Machine Learning*, Pittsburgh, PA, 2006.

Factored MDPs (FMDPs) first proposed by Boutilier et al. (1995) compactly represent the transition and reward functions of a problem using Dynamic Bayesian Networks (DBNs). Classical solution methods (i.e. DP) have been successfully adapted to manipulate such representations (Boutilier et al., 2000) and have been developed to solve large problems (Hoey et al., 1999; Guestrin et al., 2003). However, these planning techniques require a perfect knowledge of the transition and reward functions of the problem, which may not be available in practice.

Sutton and Barto (1998) describe two approaches for RL solutions when the transition and reward functions are not known. *Direct* (or *value-based*) RL algorithms build an evaluation of the optimal value function from which they build an optimal policy. Based on such RL approach, McCallum (1995) and Sallans and Hinton (2004) propose to use structured representations to handle large RL problems. *Indirect* (or *model-based*) RL algorithms build incrementally a model of the transition and reward functions. From this model, an evaluation of the optimal value function is computed using planning algorithms. Algorithms within the DYNA architecture such as DYNA-Q (Sutton, 1990) learns a tabular representation of the transition and the reward functions to incrementally update its value function and its policy at states recorded in the model.

Indirect RL methods using planning algorithms in FMDPs are described in Guestrin et al. (2002). However, these methods make strong assumptions on the problem to solve, such as knowing the structure of the problem in advance. Techniques to learn the structure of DBNs have been proposed in Chickering et al. (1997) and Friedman and Goldszmidt (1998). These methods first estimate the global structure of a DBN and then evaluate the local structure quantifying the network, which make them difficult to integrate within an incremental RL algorithm. At this time, we are not aware of any indirect RL algorithm in FMDP that does not assume at least the knowledge of the structure of

the problem to solve.

In this paper, we describe *Structured* DYNA (SDYNA), a general indirect RL framework based on FMDPs. To solve a problem with unknown reward and transition functions, SDYNA incrementally builds a structured representation of these functions and then uses adapted incremental planning algorithms from DTP research to compute a structured representation of the optimal value function. Besides, we propose to learn the reward function and the DBNs of the transition function using incremental decision tree induction algorithms (Utgoff, 1986). More precisely, we focus on SPITI, an instantiation of SDYNA which uses ITI (Utgoff et al., 1997) to learn incrementally a structured model of a stochastic RL problem. Then, a modified incremental version of the *Structured Value Iteration* (SVI) algorithm (Boutilier et al., 2000) builds a factored representation of the value function of a greedy policy from this model.

The paper is organized as follows: first, we describe SPITI within the SDYNA framework. Second, we validate our approach by comparing SPITI to DYNA-Q. We show that, unlike DYNA-Q, SPITI builds a compact representation of the problem. We also illustrate a *generalization* property resulting from the use of decision tree induction. Finally, we present some extensions and future work within the SDYNA framework.

2. Background

We first introduce some definitions used in this paper. A MDP is a tuple $\langle S, A, R, T\rangle$ where S and A are respectively a finite set of states and actions; $R : S \times A \to \mathbb{R}$ is the immediate *reward function* $R(s,a)$ and $T : S \times A \times S \to [0,1]$ is the *Markovian transition function* $P(s'|s,a)$ of the MDP. A *stationary policy* $\pi : S \times A \to [0,1]$ defines the probability $\pi(s,a)$ that the agent takes the action a in state s. The goal is to find a policy π maximizing the *value function* $V_\pi(s)$ defined using the discounted reward criterion: $V_\pi(s) = E_\pi[\sum_{t=0}^{\infty} \gamma^t \cdot r_t | s_0 = s]$, with $0 \leq \gamma < 1$ the discount factor, r_t the reward obtained at time t and s_0 the initial state, considering an infinite horizon. The action-value function $Q_\pi(s,a)$ is defined as:

$$Q_\pi(s,a) = \sum_{s' \in S} P(s'|s,a)(R(s',a) + \gamma V_\pi(s')) \quad (1)$$

The optimal value function V^* and an optimal policy π^* are defined by $V^* = V_{\pi^*}$ and $\forall \pi,\ \forall s :\ V^*(s) \geq V_\pi(s)$. Given the fully defined R and T functions, DP proposes a family of solution methods, namely *Value Iteration* and *Policy Iteration*.

2.1. Indirect Reinforcement Learning

In some problems, the reward function R and the transition model T are not known in advance. Indirect RL proposes to learn these functions by trial-and-error during the experiment. This approach is illustrated in the DYNA architecture (Sutton, 1990) that integrates *planning*, *acting* and *learning* together. The DYNA-Q algorithm is one instantiation of DYNA. In this paper, we use a stochastic version of DYNA-Q that learns a stochastic tabular representation of the transition and reward functions and then uses the update rule of the Q-learning algorithm (Watkins, 1989) weighted with the probability learned in the model to approximate V^* and π^*.

2.2. Factored Markov Decision Processes

Representing large MDPs using factored models to exploit the structure of the problem was first proposed by Boutilier et al. (1995). In a FMDP, states are decomposed as a set of random variables $S = \{X_1, \ldots, X_n\}$, where each X_i takes value in a finite domain $Dom(X_i)$. Thus, a state is defined by a vector of values $s = (x_1, \ldots, x_n)$ with $\forall i,\ x_i \in Dom(X_i)$. We denote X_i to be a variable at time t and X_i' the same variable at time $t+1$.

The state transition model T_a for an action a is composed of a transition graph represented as a DBN (Dean & Kanazawa, 1989), that is a two-layer directed acyclic graph G_a whose nodes are $\{X_1, \ldots, X_n, X_1', \ldots, X_n'\}$ (see figure 3 for an example of DBN). In G_a, the parents of X_i' are noted $Parents_a(X_i')$. In this paper, we assume that $Parents_a(X_i') \subseteq X$ (meaning that there are no *synchronic* arcs, that is arcs from X_i' to X_j'). A *Conditional Probability Distribution* $\mathrm{CPD}^a_{X_i}(X_i'|Parents_a(X_i'))$ quantifying the graph is associated to each node $X_i' \in G_a$. The full transition model of the MDP is then defined by a separate DBN model $T_a = \langle G_a, \{\mathrm{CPD}^a_{X_1}, \ldots, \mathrm{CPD}^a_{X_n}\}\rangle$ for each action a.

3. Methods

Similarly to DYNA, we propose the *Structured* DYNA (SDYNA) architecture to integrate planning, acting and learning using structured representations instead of tabular representations. This architecture aims at solving large RL problems with an unknown structure using incremental versions of planning techniques based on FMDPs.

An overview of SDYNA is given in figure 1. Fact(F) represents a factored representation of the function F. Factored representations, such as rules, decision trees

or boolean decision diagrams, allow to exploit certain regularities in F to represent or manipulate it.

Input: $Acting, Learn, Plan, \text{Fact}$
Output: $\text{Fact}(\pi)$

1. Initialization
2. At each time step t, do:
   (a) $s \leftarrow$ current (non-terminal) state
   (b) $a \leftarrow Acting(s, \{\forall a \in A\text{: Fact}(Q_{t-1}(s,a))\})$
   (c) Execute a; observe s' and r
   (d) $\{\text{Fact}(R_t), \text{Fact}(T_t)\} \leftarrow Learn(\langle s, a, s'\rangle, \text{Fact}(R_{t-1}), \text{Fact}(T_{t-1}))$
   (e) $\{\text{Fact}(V_t), \{\forall a \in A\text{: Fact}(Q_t(s,a))\}\} \leftarrow Plan(\text{Fact}(R_t), \text{Fact}(T_t), \text{Fact}(V_{t-1}))$

Figure 1. The SDYNA algorithm

The acting phase of SDYNA (steps 2.a, 2.b and 2.c) is similar to DYNA-Q and exploration policies such as ϵ-*greedy* can be used without modification. On the contrary, the learning and planning phases depend directly on the factored representation used. Unlike DYNA-Q, SDYNA does not work if the planning phase is disabled.

In the remainder, we focus on SPITI, an instantiation of SDYNA, that uses decision trees to represent the manipulated functions. The tree representation of a function F is noted $\text{Tree}(F)$. The acting phase of SPITI is instantiated by an ϵ-*greedy* exploration policy. Its learning phase (steps 2.d and 2.e in SDYNA) is based on incremental decision tree induction (section 3.1). Its planning phase (steps 2.f) is based on a modified incremental version of Structured Value Iteration (SVI), an algorithm proposed by Boutilier et al. (2000), described in section 3.2.

3.1. SPITI: Learning a Structured Model of a RL Problem

When an agent performs an action a in a state s, it can exploit two different feedbacks: its new state s' and the reward r received when doing the action a in state s. Classification algorithms learn a function from a set of examples $\langle \mathcal{A}, \varsigma \rangle$ with $\mathcal{A}$ a set of *attributes* ν_i and ς the *class* of the example. Consequently, from the observation of the agent $\langle s, a, s', r\rangle$ with $s = (x_1, \ldots, x_n)$ and $s' = (x'_1, \ldots, x'_n)$, we propose to learn the reward $R(s,a)$ and the transition $\text{CPD}^a_{X_i}$ models from the examples $\langle \mathcal{A} = (x_1, \ldots, x_n, a), \varsigma = r\rangle$ and $\langle \mathcal{A} = (x_1, \ldots, x_n), \varsigma = x'_i\rangle$, respectively.

Trials of the agent in the environment at each time step compose a stream that must be learned incrementally. Decision tree induction research propose incremental algorithms (Utgoff, 1986) able to build a decision tree $\text{Tree}(F)$ representing a factored representation of F from a stream of examples $\langle \mathcal{A}, \varsigma \rangle$.

Moreover, using decision tree induction allows the agent to *generalize* from its history. Actually, visiting all the possible state/action pairs is infeasible in large problems. Unlike tabular representations, decision trees are able to propose a default class distribution for examples that have not been presented. Consequently, by generalizing from its history, an agent may be able to choose an adequate action even in states not visited yet.

In SPITI, we use the decision tree induction algorithm named ITI (Utgoff et al., 1997) and noted $ITI(\text{Tree}(F), \mathcal{A}, \varsigma)$. We refer to Utgoff et al. (1997) for a complete description of ITI.

Input: $s, a, s', \text{Tree}(R_{t-1}), \text{Tree}(T_{t-1})$
Output: $\{\text{Tree}(R_t), \text{Tree}(T_t)\}$

1. $\mathcal{A} \leftarrow \{x_1, \ldots, x_n\}$
2. $\text{Tree}(R_t) \leftarrow ITI(\text{Tree}(R_{t-1}), \mathcal{A} \bigcup \{a\}, r)$
3. $\text{Tree}(T_t) \leftarrow \text{Tree}(T_{t-1})$
4. For all $i \in |X|$:
   $\text{Tree}(\text{CPD}^a_{X_i} \in T_t) \leftarrow ITI(\text{Tree}(\text{CPD}^a_{X_i} \in T_{t-1}), \mathcal{A}, x'_i)$
5. Return $\{\text{Tree}(R_t), \text{Tree}(T_t)\}$

Figure 2. SPITI (1): the *Learn* algorithm.

Figure 2 describes the *Learn* algorithm in SPITI. As shown in step 2, the reward learning algorithm is straightforward. An example is composed of the vector $s = (x_1, \ldots, x_n)$ and the action a, while its class ς is defined as the reward r received by the agent. The example is then learned by ITI that builds a model of the reward function as a tree with the leaves labeled with the values $r \in \mathbb{R}$.

The transition model T_a of the action a is composed of the graph G_a and the set of $\text{CPD}^a = (\text{CPD}^a_{X_1}, \ldots, \text{CPD}^a_{X_n})$. As illustrated in step 4, we propose to learn separately each $\text{CPD}^a_{X_i}$ as a decision tree without trying to build the complete DBN G_a. One tree is used to represent each $\text{CPD}^a_{X_i}$. This tree is built by learning examples composed of the vector $s = \{x_1, \ldots, x_n\}$ while the class ς is defined as the instantiation of X'_i in the state s' of the system[1]. This method is justified because we suppose no synchronic

[1]Instead of having a different $\text{Tree}(\text{CPD}^a_{X_i})$ for each action and for each variable, one may maintain only one $\text{Tree}(CPD_{X_i})$ for each variable by adding the action a to the set of attributes $\mathcal{A}$. We did not consider this case in

arc (thus, we have $X'_i \perp\!\!\!\perp X'_j \,|\, X_1, \ldots, X_n$).

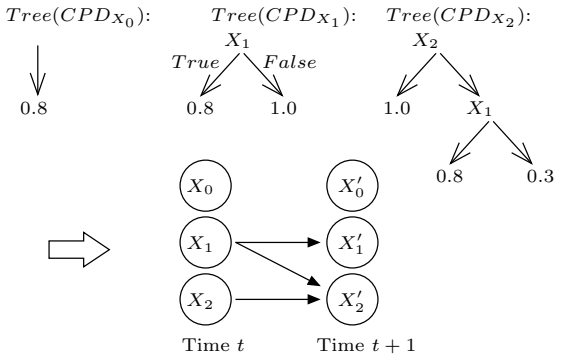

Figure 3. Structure of a DBN G from a set of decision trees $\{\text{Tree}(\text{CPD}_{X_i})\}$. In $\text{Tree}(\text{CPD}_{X_2})$, the leaf labeled 0.3 means that the probability for X'_2 to be true is $P(X'_2|X_1 = False, X_2 = False) = 0.3$.

The probability $\text{CPD}^a_{X_i}(X'_i|Parents_a(X'_i))$ is computed at each leaf of $\text{Tree}(\text{CPD}^a_{X_i})$ from the training examples present at this leaf. The SDYNA framework does not require to explicitly build the global structure of the DBNs. However, one can build each DBN G_a by assigning to $Parents_a(X'_i)$ the set of variables X_i used in the tests in each $\text{Tree}(\text{CPD}^a_{X_i})$ (see figure 3 for an example).

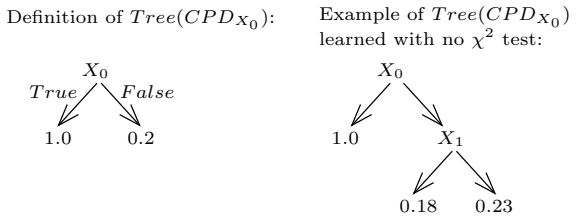

Figure 4. A test of independence between two probability distributions is required to avoid unnecessary tests in the tree (as shown with the X_1 variable test). Such a test is not required for deterministic transitions (i.e. the leaf containing the probabillity of 1.0).

To determine the best test to install at a decision node, decision tree induction algorithms are based on an information-theoric metric. This metric is computed for each attribute $\nu_i \in \mathcal{A}$. Different metrics have been proposed such as *gain ratio*, *Kolmogorov-Smirnoff* or χ^2. In SPITI, we use χ^2 because the metric can also be used as a test of independence between two probability distributions, as suggested by Quinlan (1986).

Indeed, an action a executed in similar states $s = \{x_1, \ldots, x_n\}$ may have different effects in stochastic problems. When learning $\text{Tree}(\text{CPD}^a_{X_i})$, examples with similar set of attributes but different class must be classified at the same leaf because they follow the same probability distribution. In figure 4, we define $P(X'_0|X_0 = False) = 0.2$. Thus, the examples $\langle\{X_0 = False, X_1 = False\}, X'_0 = True\rangle$ and $\langle\{X_0 = False, X_1 = True\}, X'_0 = False\rangle$ must be classified in the same leaf. In SPITI, we use a χ^2 test to check the independence between two probability distributions ($P(X'_0|X_0 = False, X_1 = False)$ and $P(X'_0|X_0 = False, X_1 = True)$ in figure 4). Thus, a decision node is installed only if the χ^2 value associated with the test to install is above a threshold, noted τ_{χ^2}. The problem does not occur for deterministic transitions.

3.2. SPITI: Factored Planning

The planning algorithm in SPITI is a modified incremental version of the Structured Value Iteration (SVI) algorithm proposed by Boutilier et al. (2000). SVI is adapted from the Value Iteration algorithm using decision trees as factored representation. It is described in figure 5.

Input: $\langle G_a|P_a\rangle$, $\text{Tree}(R)$ Output: $\text{Tree}(\pi^*)$

1. $\text{Tree}(V) \leftarrow \text{Tree}(R)$
2. Repeat until termination:
   (a) $\forall a \in A : \text{Tree}(Q^V_a) \leftarrow Regress(\text{Tree}(V), a)$
   (b) $\text{Tree}(V) \leftarrow Merge(\{\text{Tree}(Q^V_a) : \forall a \in A\})$ (using maximization over the value as combination function).
3. $\forall a \in A : \text{Tree}(Q^V_a) \leftarrow Regress(\text{Tree}(V), a)$
4. $\text{Tree}(\pi) \leftarrow Merge(\{\text{Tree}(Q^V_a) : \forall a \in A\})$ (using maximization over the value as combination function and placing the action as label of the leaf).
5. Return $\text{Tree}(\pi)$

Figure 5. The Structured Value Iteration (SVI) algorithm from Boutilier et al. (2000).

SVI uses two operators. First, $Regress(\text{Tree}(V), a)$ produces the action-value function $\text{Tree}(Q^V_a)$ representing the expected value of performing the action a with respect to the value function $\text{Tree}(V)$. Thus, step 2.a and 3 compute eq. (1) for action a and for all states s using tree representations.

Second, $Merge(\{T_1, \ldots, T_n\})$ produces a single tree containing all the partitions occurring in all the trees $T_1, \ldots, T_n$ to be merged, and whose leaves are labeled using a *combination function* of the labels of

this paper.

the corresponding leaves in the original trees[2]. In step 2.b and 4, the value function Tree(V) and the policy Tree(π) respectively, are built by selecting for all the partitions occuring in all the trees the action with the best associated value Tree(Q_a^V). We refer to Boutilier et al. (2000) for a comprehensive description of both *Merge* and *Regress* operators.

Using SVI without modifications in SPITI is feasible but not practical for two reasons. First, SVI would improve the value function until convergence despite an incomplete model. Second, the output of SVI is a greedy policy which may not be adapted in a problem where the transition and reward functions are unknown. In such problems, the agent needs to explore by trying actions that may not be optimal given its current knowledge. Thus, we propose a modified version of SVI in SPITI as described in figure 6.

Input: Tree(R_t), Tree(T_t), Tree(V_{t-1})
Output: Tree(V_t), $\{\forall a \in A$: Tree($Q_t(s,a)$)$\}$

1. $\forall a \in A$: Tree($Q_t(s,a)$) $\leftarrow$ *Regress*(Tree(V_{t-1}), a)
2. Tree(V_t) $\leftarrow$ *Merge*($\{$Tree($Q_t(s,a)$) : $\forall a \in A\}$) (using maximization over the value as combination function).
3. Return $\{$Tree(V_t), $\{\forall a \in A$: Tree($Q_t(s,a)$)$\}\}$

Figure 6. SPITI (3): the *Plan* algorithm based on SVI.

At each time step, the *Plan* algorithm computes one iteration of SVI and updates the mapping from an action a to a structured representation of its corresponding action-value function Tree($Q_t(s,a)$). Then, this mapping is used in the acting phase by the exploration policy, i.e. ϵ-*greedy* or softmax action selection (Sutton & Barto, 1998). The greedy policy is built by selecting the best action in state s according to the expected values returned by Tree($Q_t(s,a)$).

The value function Tree(V_t) associated to the greedy policy is also computed by merging the set of action-value functions $\{$Tree($Q_t(s,a)$) : $\forall a \in A\}$ (using maximization as combination function). Tree(V_t) is then reused at time $t+1$ to compute the action-value functions Tree($Q_{t+1}(s,a)$). If the transition model and the reward function learned by the agent are stationary, Tree(V_t) converges in a finite number of time steps to the optimal value function Tree(V^*) given the current knowledge of the agent (Boutilier et al., 2000). However, as long as the model of the agent is not sufficiently accurate, Tree(V^*) may be significantly different from the optimal value function of the RL problem to solve.

[2]The combination function used during the merge process is noted in the description of the algorithm using it.

4. Results

Empirical evaluations of SPITI have been run on two problems[3], namely *Coffee Robot* and *Process Planning*, defined in (Boutilier et al., 2000). For each problem, we compared SPITI to DYNA-Q. For both algorithms, we used $\gamma = 0.9$ and the ϵ-*greedy* exploration policy with a fixed $\epsilon = 0.1$. In DYNA-Q, we used $\alpha = 1.0$ and set N equals to twice the size of the model. The function $Q(s,a)$ is initialized optimistically. In SPITI, the threshold τ_{χ^2} used in the χ^2 test to detect the independence between two distributions was set to 10.6 (corresponding to a probability of independence superior to 0.99).

We also ran two agents, noted RANDOM and OPTIMAL, executing at each time step, respectively, a random action and the best action. The RANDOM agent learns a transition and reward models using *LearnR* and *LearnT* algorithms from SPITI. The policy of OPTIMAL has been computed off-line using SVI with the span semi-norm as a termination criterion using a threshold of 0.01. The transition and the reward model of the OPTIMAL agent are the static decision trees defining the problem.

To allow the agents to perform multiple trials in the same problem, a set of initial states and a set of terminal states are added to the problem definitions. As soon as an agent is in a terminal state, its new state is then initialized randomly in one of the possible initial states using a uniform distribution (whatever the action executed by the agent). Each agent is evaluated T time steps during an experiment.

We specifically focused on the *discounted reward* R^{disc} obtained by the agent and the size of its transition model built from its history. R_t^{disc} at time t is defined as follows: $R_t^{disc} = r_t + \gamma' R_{t-1}^{disc}$ with r_t the reward received by the agent and $\gamma' = 0.99$[4]. We do not show results concerning the size of the reward function because they are similar to the results concerning the size of the transition model. Moreover, we did not focus on the size of the value function computed or the time required for executions because Boutilier et al. (2000) provide exhaustive results on this topic.

4.1. Coffee Robot Problem

The *Coffee Robot* problem[5] is a small stochastic problem where a robot has to go to a café to buy some

[3]Their complete definitions can be found on the SPUDD website http://www.cs.ubc.ca/spider/jhoey/spudd/spudd.html.

[4]$\gamma' = 0.99$ makes the results more readable.

[5]This problem is labeled as "*Coffee*" on the SPUDD website.

coffee and return it to its owner. The problem is composed of 4 actions and 6 boolean variables (defining $2^6 * 4 = 256$ state/action pairs). The terminal states are states where the user has a coffee. The set of initial states is composed of all non-terminal states. We run 30 experiments for each agent with $T = 4000$.

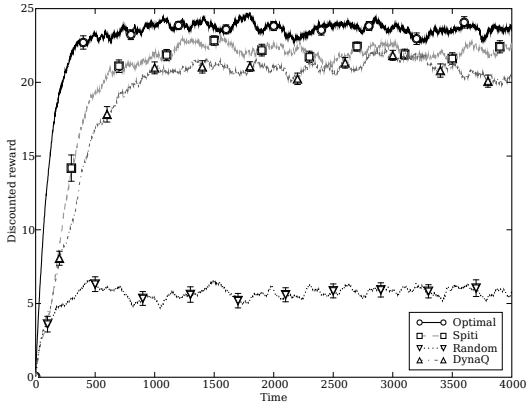

Figure 7. Discounted reward obtained from the *Coffee Robot* problem.

Figure 7 shows the discounted reward obtained by each agent. Both DYNA-Q and SPITI behave similarly on a small problem and quickly execute a near optimal policy (in approximately 1000 time steps). However, they do not execute an optimal policy because of the *ϵ-greedy* exploration policy.

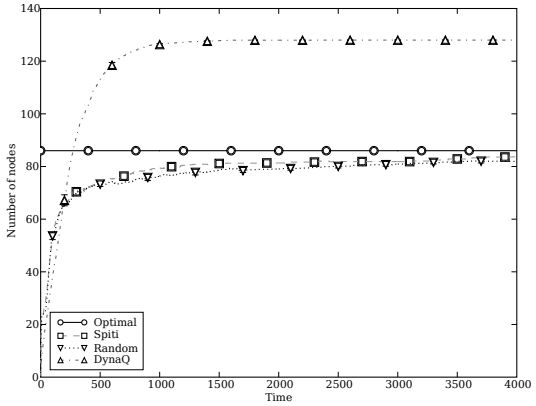

Figure 8. Size of the transition model learned from the *Coffee Robot* problem.

Figure 8 compares the different sizes of the transition model learned by SPITI, DYNA-Q and RANDOM. The number of nodes of the model built by DYNA-Q is equal to the number of transitions in the problem (128 entries because a terminal state does not have transitions). The model learned by SPITI and RANDOM is significantly smaller (less than 90 nodes) because they build a structured representation of the problem. The discounted reward obtained by SPITI (figure 7) shows that this representation is exploited by the planning algorithm to compute incrementally a near optimal policy. Moreover, because they are based on the same learning methods, the transition model learned by RANDOM and SPITI are similar in size despite their different policies.

4.2. Process Planning Problem

The *Process Planning* problem[6] is a stochastic problem from a manufacturing domain composed of 14 actions and 17 binary variables (defining $2^{17} * 14 = 1,835,008$ state/action pairs). A product must be achieved by attaching two manufactured components together. High quality components can be produced by using actions such as hand-paint or drill; low quality components can be produced by using actions such as spray-paint or glue. The agent must produce low or high quality components depending on the demand. Terminal states are states where two components are attached together. Initial states are all non-terminal states from which there is at least one policy to get to a terminal state. We run 20 experiments for each agent with $T = 10000$.

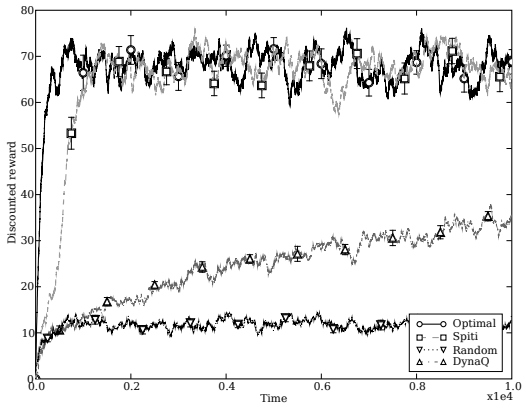

Figure 9. Discounted reward obtained on the *Process Planning* problem.

Figure 9 shows the discounted reward obtained by each agent. SPITI executes a near optimal policy in approximately 2000 time steps unlike DYNA-Q which, after 10000 steps, is still improving its policy. This differ-

[6]This problem is labeled as "*Factory*" on the SPUDD website.

ence illustrates the generalization property of decision tree induction algorithms. From the agent history, SPITI is able to compute a policy which may propose a good action for states that have not been visited yet. On the contrary, DYNA-Q must try all the state/action pairs before considering them during planning.

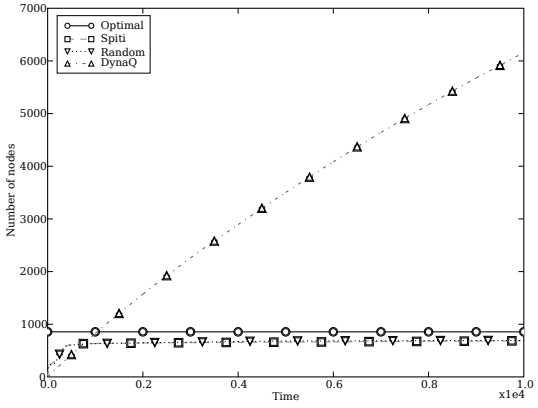

Figure 10. Size of the transition model learn in the *Process Planning* problem.

Figure 10 shows the size of the transition model learned by the agents. Similarly to the *Coffee Robot* problem, the space requirements of the tabular representation of DYNA-Q grow exponentially with the number of variables and actions of the problem. On the contrary, SPITI quickly builds a compact representation of the problem sufficiently accurate to perform a policy similar to OPTIMAL (figure 9).

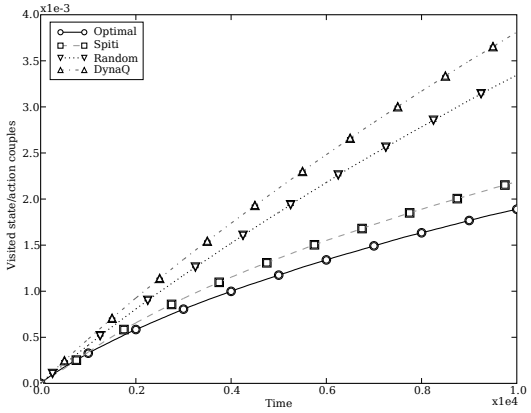

Figure 11. Proportion of state/action pairs visited on the total number of pairs existing in the *Process Planning* problem

Figure 11 shows the number of state/action pairs visited by each agent. It illustrates the intensive exploration of RANDOM and DYNA-Q in large problem compared to SPITI or OPTIMAL. Unlike DYNA-Q and despite an important number of state/action pairs, the model learned by RANDOM is similar in size to SPITI showing that the size of the model mainly depends on the structure of the problem and not on the number of visited state/action pairs.

5. Discussion

FMDP methods exploit a given exhaustive and structured representation to compute an optimal or near-optimal policy in large problems. The first contribution of this paper was to propose SDYNA, a indirect RL framework able to use powerful planning techniques in structured RL problem where the reward and transition functions are unknown. We described an instantiation of SDYNA, called SPITI, combining a modified version of SVI with ITI, a decision tree induction algorithm used to learn the structure of a RL problem. We showed that SPITI performs better than DYNA-Q, an indirect RL algorithm using tabular representations. Further work is still necessary to compare the SDYNA framework to direct RL approach based on factored representations such as McCallum (1995).

Some alternative planning algorithms to SVI have been proposed, such as SPUDD (St-Aubin et al., 2000) or techniques based on Linear Programming algorithms (Guestrin et al., 2003). These techniques have been shown to perform better in speed or memory than SVI. Our incremental version of SVI was the most straightforward planning method to validate our approach. We are currently working on integrating some of these more efficient planning methods in SDYNA.

These planning methods would not fundamentally change the results presented in this paper. As we have shown above, the size of the learned model does not strictly depend on the policy of the agent. Moreover, given a model at time t, two greedy policies computed by two different planning algorithms would have similar value functions. However, using more efficient planning methods would allow SPITI to address larger RL problems.

The second contribution of this paper is to illustrate the generalization property of using decision tree induction algorithms within the FMDP framework. In SPITI, this property depends on the value of the threshold τ_{χ^2} used in the χ^2 test to detect the independence between two distributions. This parameter drives the decision node creation process in ITI between two extrema (in a fully stochastic problem): $Parents_a(X'_i) =$

$\emptyset$ and $Parents_a(X'_i) = \{\forall i, X_i\}$. In the empirical studies described in this paper, we set $\tau_{\chi^2} = 10.6$ to split two distributions when the probability of independence returned by the χ^2 test was superior to 0.99. We believe that further work is necessary to analyze the dependencies between the value of τ_{χ^2}, the size of the learned model and the consequences on the quality of the policy computed during the planning phase.

Furthermore, the generalisation property is necessary for designing exploration policies in large problems because it may render an exhaustive exploration unnecessary, as we have shown above with the *ϵ-greedy* exploration policy. Recent research on the exploration/exploitation tradeoff has yielded more efficient algorithms such as *Factored* E^3 or *Factored* R_{max} (Guestrin et al., 2002) with relevant theoretical guarantees but with strong assumptions on the FMDP to solve. Thus, further work will be necessary to integrate these algorithms in SDYNA to handle the exploration/exploitation dilemma while the structure of the FMDP is built incrementally.

6. Conclusion

In this paper, we have presented SDYNA, an architecture designed to integrate planning, acting and learning in the context of FMDPs. Through one instantiation of SDYNA called SPITI, we have shown that this architecture can build a compact representation of the model of a RL problem with an unknown structure by using decision tree induction algorithms. Furthermore, we have shown that for a given set of visited states, the generalization property of this method results in a faster policy improvements than methods using tabular representations.

Acknowledgement

Thanks to the anonymous referees for their suggestions. We also wish to thank Christophe Marsala and Vincent Corruble for useful discussions.

References

Boutilier, C., Dearden, R., & Goldszmidt, M. (1995). Exploiting Structure in Policy Construction. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)* (pp. 1104–1111). Montreal.

Boutilier, C., Dearden, R., & Goldszmidt, M. (2000). Stochastic Dynamic Programming with Factored Representations. *Artificial Intelligence*, *121*, 49–107.

Chickering, D. M., Heckerman, D., & Meek, C. (1997). A Bayesian Approach to Learning Bayesian Networks with Local Structure. *Proceedings of the 13th International Conference on Uncertainty in Artificial Intelligence* (pp. 80–89).

Dean, T., & Kanazawa, K. (1989). A Model for Reasoning about Persistence and Causation. *Computational Intelligence*, *5*, 142–150.

Friedman, N., & Goldszmidt, M. (1998). Learning Bayesian Networks with Local Structure. *Learning and Inference in Graphical Models*. M. I. Jordan ed.

Guestrin, C., Koller, D., Parr, R., & Venkataraman, S. (2003). Efficient Solution Algorithms for Factored MDPs. *Journal of Artificial Intelligence Research*, *19*, 399–468.

Guestrin, C., Patrascu, R., & Schuurmans, D. (2002). Algorithm-Directed Exploration for Model-Based Reinforcement Learning in Factored MDPs. *ICML-2002 The Nineteenth International Conference on Machine Learning* (pp. 235–242).

Hoey, J., St-Aubin, R., Hu, A., & Boutilier, C. (1999). SPUDD: Stochastic Planning using Decision Diagrams. *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence* (pp. 279–288). Morgan Kaufmann.

McCallum, A. K. (1995). *Reinforcement Learning with Selective Perception and Hidden State*. Doctoral dissertation, Department of Computer Science, University of Rochester, USA.

Quinlan, J. R. (1986). Induction of Decision Trees. *Machine Learning*, *1*, 81–106.

Sallans, B., & Hinton, G. E. (2004). Reinforcement Learning with Factored States and Actions. *Journal of Machine Learning Research*, *5*, 1063–1088.

St-Aubin, R., Hoey, J., & Boutilier, C. (2000). APRICODD: Approximate Policy Construction Using Decision Diagrams. *NIPS* (pp. 1089–1095).

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *Proceedings of the Seventh International Conference on Machine Learning* (pp. 216–224). San Mateo, CA. Morgan Kaufmann.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Utgoff, P. (1986). Incremental Induction of Decision Trees. *Machine Learning*, *4*, 161–186.

Utgoff, P. E., Nerkman, N. C., & Clouse, J. A. (1997). Decision Tree Induction Based on Efficient Tree Restructuring. *Machine Learning*, *29*, 5–44.

Watkins, C. J. C. H. (1989). *Learning with Delayed Rewards*. Doctoral dissertation, Psychology Department, University of Cambridge, England.