Exploiting Additive Structure in Factored MDPs for Reinforcement Learning

Thomas Degris, Olivier Sigaud, and Pierre-Henri Wuillemin

Université Pierre et Marie Curie - Paris6
4 place Jussieu, F-75005 Paris, France
thomas.degris@laposte.net, Olivier.Sigaud@lip6.fr,
Pierre-Henri.Wuillemin@lip6.fr

Abstract. SDYNA is a framework able to address large, discrete and stochastic reinforcement learning problems. It incrementally learns a FMDP representing the problem to solve while using FMDP planning techniques to build an efficient policy. SPITI, an instantiation of SDYNA, uses a planning method based on dynamic programming which cannot exploit the additive structure of a FMDP. In this paper, we present two new instantiations of SDYNA, namely ULP and UNATLP, using a linear programming based planning method that can exploit the additive structure of a FMDP and address problems out of reach of SPITI.

1 Introduction

Markov Decision Processes (MDPs) are a fundamental framework to model planning under uncertainty problems as well as Reinforcement Learning (RL) problems. Standard exact solution methods for both problems are known to work well but are inappropriate for large problems because they require explicit state space enumerations. Among different approximation techniques, factored MDPs (FMDPs), first proposed by [1], assume the decomposition of the state space with random variables. FMDPs utilize dependencies between variables, defined using Dynamic Bayesian Networks (DBNs) [2], to compactly represent the transition and reward functions of structured MDPs.

When the structure of the transition and reward functions are fully known, solution methods may be used to compute optimal (or near optimal) value functions and policies of the FMDP. These solution methods are based on two different classical techniques, namely Dynamic Programming (DP) and Linear Programming (LP). First, algorithms such as Structured Policy Iteration (SPI), Structured Value Iteration (SVI) and Stochastic Planning Using Decision Diagrams (SPUDD) are based on DP [3,4] and mainly exploit context specific independence by using structured representations (i.e. decision trees or decision diagrams) to manipulate the functions of the FMDPs. Second, [5] proposes different algorithms based on LP that can exploit additional regularities such as the additive decomposition of the reward function and an additive approximation of the value function. We name such regularities as additive structure of the problem.

S. Girgin et al. (Eds.): EWRL 2008, LNAI 5323, pp. 15–26, 2008.

When the structure of the transition and reward functions are unknown, [6] has proposed Structured DYNA (SDYNA). SDYNA is a general framework combining supervised learning algorithms with planning methods to solve large, discrete and stochastic RL problems. SPITI is an instance of SDYNA that uses an incremental decision tree induction algorithm combined with an incremental version of SVI. Consequently, SPITI is not able to exploit additive structure of FMDPs and is very limited to represent and solve problems such as the SysAdmin problem [5].

In this paper, we describe two new instances of SDYNA, namely ULP and UNATLP. Both instances use LP based planning method to exploit the additive structure of the problem. Moreover, we propose in UNATLP a different representation of the transition function in the FMDP that is learned. First, we show that both ULP and UNATLP are able to exploit the additive structure of a problem without knowing its structure in advance, allowing these instances to address a RL problem with $4 \cdot 10^{13}$ state/action pairs. At this time, we are not aware of any model-based RL algorithm able to solve such large problems without assuming the knowledge of its structure. Second, we show that the new representation of the FMDP in UNATLP outperforms the one used in SPITI and ULP on such problems.

The remainder of this paper is organized as follows: in Section 2, we introduce FMDPs, the planning method based on LP proposed by [5] and SDYNA. In Section 3, we describe ULP and UNATLP. Section 4 describes and discusses empirical results of these instances on the SysAdmin problem.

2 Background

We first introduce some definitions used in this paper. A MDP is defined by a tuple $\langle S, A, R, P \rangle$ where S is a finite set of states, A is a finite set of actions, R is the immediate *reward function* with $R : S \times A \to \mathbb{R}$ and P is the *Markovian transition function* $P(s'|s, a)$ with $P : S \times A \times S \to [0, 1]$. A *stationary policy* π is a mapping $S \to A$ with $\pi(s)$ defining the action to be taken in state s.

We evaluate a policy π in state s, considering an infinite horizon, with 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 and r_t the reward obtained at time t. A policy π is optimal if $\forall s \in S, \forall \pi' : V_\pi(s) > V_{\pi'}(s)$. The value function of an optimal policy π^* is called *optimal value function* and is noted V^*.

The action-value function $Q_a^V(s)$ for an action a and a value function $V(s)$ is defined as $Q_a^V(s) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a)V(s')$. For a given value function V, it is possible to define a greedy policy relative to V, noted Greedy_V, by taking for each state s the action with the best action-value:

$$\text{Greedy}_V(s) = \arg\max_a [R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a)V(s')] \tag{1}$$

The greedy policy relative to V^* is an optimal policy $\pi^*(s) = \text{Greedy}_{V^*}(s)$.

We now assume that states are composed of a set of random variables $X = \{X_1, \ldots, X_n\}$. A state is then defined by a vector $s = (x_1, \ldots, x_n)$ with $\forall i, x_i \in \text{Dom}(X_i)$. FMDPs are a framework exploiting the structure of the problem to represent compactly large MDPs [1]. For each action a, the transition model of the FMDP is defined by a separate DBN model $T_a = \langle G_a, \{P^a_{X_1}, \ldots, P^a_{X_n}\}\rangle$. G_a is a two-layer directed acyclic graph whose nodes are $\{X_1, \ldots, X_n, X'_1, \ldots, X'_n\}$ with X_i a variable at time t and X'_i the same variable at time $t+1$. The parents of X'_i are noted $\text{Parents}_a(X'_i)$ with $\text{Parents}_a(X'_i) \subseteq X$. The transition model T_a is quantified by *Conditional Probability Distributions* (CPDs), noted $P^a_{X_i}(X'_i|\text{Parents}_a(X'_i))$, associated to each node $X'_i \in G_a$.

A similar decomposition provides a compact representation of the reward function. First, we formalize the concept of *localized* function [5]: a function f has a *scope* $\text{Scope}(f) = Y \subseteq X$ if $f : \text{Dom}(Y) \mapsto \mathbb{R}$. We use $f(x)$ as shorthand for $f(y)$ where y is the part of the instantiation x corresponding to variables in Y. The reward function may now be defined as the sum $\sum_{j=1}^{r} R_j(s) \in \mathbb{R}$ where each function R_j is a localized function with $\text{Scope}(R_j)$ restricted to a small set of variables. There may be a different decomposition R^a_j for each action a.

2.1 Linear Programming Based Approximation in MDPs

Different approaches may be used to compute the optimal policy in a MDP. One approach is to represent the MDP as a linear program (LP) [7]. Such LP is defined as:

For variables: $V(s)$, $\forall s \in S$
Minimize: $\sum_s \alpha(s)V(s)$
Subject to: $V(s) \geq R(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s')\forall s \in S, \forall a \in A$ (LP 1)

where $\alpha(s)$ are the state relevance weights ($\alpha(s) > 0$ for each state s). One major issue that prevents this LP from being applied to real problems is that it uses explicit enumerations of the state space which is usually very large. The remaining of this section presents previous work from [5] to avoid such enumerations.

One approach to address large state space is to approximate value functions with *linear value function* $\sum_{i=1}^{k} w_i h_i(s)$ for some coefficients $(w_1, \ldots, w_k)$. The set $\{h_1, \ldots, h_k\}$ is a set of *basis functions* where each h_i is a localized function, defining the space $\mathcal{H}$ of allowable value functions. We discuss the choice of such functions in section 5. Replacing explicit state value function $V(s)$ by the approximation, (LP 1) is rewritten to produce an approximation of the optimal value function of the MDP in $\mathcal{H}$ [8]:

For variables: $w_1, \ldots, w_k$
Minimize: $\sum_s \alpha(s) \sum_{i=1}^{k} w_i h_i(s)$
Subject to: $\sum_{i=1}^{k} w_i h_i(s) \geq R(s,a) +$
$\gamma \sum_{s'} P(s'|s,a) \sum_{i=1}^{k} w_i h_i(s')\forall s \in S, \forall a \in A$ (LP 2)

This linear program is guaranteed to be feasible if a constant function h_0 is included in the set of basis functions. We assume that this is the case in the remaining of the paper.

(LP 2) reduces the number of free variables from $|S|$ to k. However, the number of constraints remains $|S|\times|A|$, each constraint is potentially a sum of $|S|$ terms, and the objective function is still a sum of $|S|$ terms. We now describe an outline of the method proposed by [5] to represent this linear program compactly, exploiting the structure of the problem.

2.2 Linear Programming Based Approximation in Factored MDPs

The compact representation of (LP 2) is based on a *factored* linear value function representation [9], that is a linear value function over the basis $h_1, \ldots, h_k$ where each localized function h_i is restricted to a small number of state variables. The different restricted domain functions defined in the FMDP result in efficient computations on compact representations over large state spaces. The first operation is to compute the backprojection $g_i^a(s) = \sum_{s'} P(s'|s,a)h_i(s')$ of a basis function h_i for an action a and given the graph G_a. Recalling that the scope of a basis function h_i is small, the backprojection may be simplified, as shown in [9]. The scope of $g_i^a(s)$ is $\text{Scope}(g_i^a) = \cup_{X'_j \in \text{Scope}(h_i)} \text{Parents}_a(X'_j)$. Consequently, the cost of the computation depends linearly on $|\text{Dom}(\text{Scope}(g_i^a))|$, which depends on the scope of h_i and the complexity of the graph G_a.

The state relevance weights $\alpha(s)$ in the objective function of (LP 2) may be considered as distribution over states, so that $\alpha(s) > 0$ and $\sum_s \alpha(s) = 1$. As suggested by [5], we use uniform state relevance weights defined as $\alpha(s) = \frac{1}{|S|}$. Using a different reorganisation of the set of constraints and the objective function, a new linear program may be formulated [5]:

For variables: $w_1, \ldots, w_k$
Minimize: $\sum_{i=1}^{k} w_i \alpha_i$
Subject to: $0 \geq \sum_{i=1}^{k} w_i [\gamma g_i^a(s) - h_i(s)] + \sum_{j=1}^{r} R_j^a(s) \ \forall s \in S, \forall a \in A$ (LP 3)

where $\alpha_i = \sum_{c_i \in \text{Dom}(C_i)} \alpha(c_i) h_i(c_i)$ with $C_i = \text{Scope}(h_i)$ and $\alpha(c_i)$ the marginal of the state relevance weights α over C_i.

(LP 3) has now only k free variables, k terms in the objective function and each constraint may be computed on a restricted scope. However, the number of constraints is still exponential. As described by [5], these constraints may be represented compactly using a variable elimination algorithm. We note FactoredALP the algorithm building (LP 3) and returning the solution. We refer to [5] for a comprehensive description of this algorithm.

2.3 Context-Specific Independence

Similarly to solution methods in FMDP such as SPI, SVI and SPUDD [3,4], [5] exploits context-specific independence, using a rule-based representation [10], to compute the backprojections and to reduce the number of constraints in (LP 3). Entries with the same value of a function are referred to as *consistent* contexts. Two contexts $c \in \text{Dom}(C)$ and $b \in \text{Dom}(B)$ with $C \subseteq \{X, X'\}$ and $B \subseteq \{X, X'\}$ are defined as consistent if they have the same assignment for the variables in $C \cap B$.

A *probability rule* $\eta = |c : p|$ is a function $\eta : \{X, X'_i\} \mapsto [0, 1]$, where the context $c \in \text{Dom}(C)$, $C \subseteq \{X, X'_i\}$ and $p \in [0, 1]$ and such that $\eta(x, x'_i) = p$ if x and x'_i are consistent with c and $\eta(x, x'_i) = 1$ otherwise. A *rule-based CPD* is a function $P_a : (\{X'_i\} \cup X) \mapsto [0, 1]$ composed of a set of probability rules $\{\eta_1, \ldots, \eta_n\}$ whose contexts are mutually exclusive and exhaustive. A *value rule* $\rho = |c : v|$ is a function $\rho : X \to \mathbb{R}$ such that $\rho(s) = v$ when s is consistent with c and 0 otherwise. A *rule-based function* $f : X \mapsto \mathbb{R}$ is composed of a set of rules $\{\rho_1, \ldots, \rho_n\}$ such that $f(x) = \sum_{i=1}^{n} \rho_i(x)$. Both reward and basis functions may be represented as a rule-based function.

The transition, reward and value functions in a FMDP may be represented using rule-based representations. [5] utilizes this representation to rewrite the computation of the backprojections and to represent (LP 3) compactly, exploiting context-specific independence. We refer to their paper for a complete description.

2.4 Structured DYNA and Spiti

The solution method based on LP described in the previous section assumes that the structure of the problem is available, which may not be the case in practice. Recently, [6] has proposed *Structured* DYNA (SDYNA), that is a framework able to learn the structure of a FMDP from experience. SDYNA is described in Figure 1, where $\mathsf{Fact}[F]$ represents a factored representation of a function F.

Input: Acting, Learn, Plan, Fact Output: $\emptyset$

1. Initialize the FMDP $\mathcal{F}_0$
2. At each time step t, with s the current (non-terminal) state, do:
   (a) $a \leftarrow \text{Acting}(s, \{\mathsf{Fact}[Q^a_{t-1}], \forall a \in A\})$
   (b) Execute a; observe s' and r
   (c) $\mathcal{F}_t \leftarrow \text{Learn}(\mathcal{F}_{t-1}, \langle s, a, s', r\rangle)$
   (d) $\{\mathsf{Fact}[V_t], \{\mathsf{Fact}[Q^a_t], \forall a \in A\}\} \leftarrow \text{Plan}(\mathcal{F}_t, \mathsf{Fact}[V_{t-1}])$

Fig. 1. The SDYNA algorithm

SDYNA is decomposed in three phases. First, from its current policy, represented as the set $\{\mathsf{Fact}[Q^a_{t-1}], \forall a \in A\}$ of action-value function, an action is executed during the acting phase (step 2.a, 2.b and 2.c). Second, from the observation $\langle s, a, s', r\rangle$, the FMDP $\mathcal{F}$ representing a model of the problem is updated during the learning phase (step 2.d). Finally, the set $\{\mathsf{Fact}[Q^a_{t-1}], \forall a \in A\}$ of action-value functions is updated during the planning phase (step 2.e).

SPITI is an instantiation of SDYNA, also presented in [6], that incrementally learns structured representations of the transition and reward functions using an induction of decision tree algorithm, noted $\text{UpdateTree}(\mathsf{Tree}[F], x, y)$ with $\mathsf{Tree}[F]$ the decision tree representation of the function F to update, x the input of F and y its output. First, for the action a of the observation, each CPD

$\mathsf{Tree}[P^a_{X_i}]$ is updated with UpdateTree($\mathsf{Tree}[P^a_{X_i}], (x_1, \ldots, x_n), x'_i$) using the state $s = (x_1, \ldots, x_n)$ as the set of inputs and the value of the variable X_i in s' as output. Second, the reward function $\mathsf{Tree}[R]$ is updated using the current state and action as input and the reward observed as output. SPITI uses χ^2 as its information-theoric metric. Moreover, a decision node in a tree $\mathsf{Tree}[P^a_{X_i}]$ is installed only if the χ^2 value for the variable is above a threshold τ_{χ^2} [11]. SPITI uses an incremental version of the SVI algorithm [3] during the planning phase to update the set $\{\mathsf{Tree}[Q^a_{t-1}], \forall a \in A\}$ of action-value functions. All the functions of a FMDP, that is the CPDs in the transition function, the reward function and the value function, are represented with decision trees in SPITI, as in SVI.

3 Exploiting Additive Structure in SDYNA

SPITI suffers from two strong limitations to be able to exploit the additive structure of a RL problem. First, the reward function is represented with one tree $\mathsf{Tree}[R]$ which is not adapted to represent functions with an additive structure. Second, it plans with an incremental version of SVI, which is not able to exploit the additive structure of a problem and performs very poorly on such problems [3,5]. We address both issues in the remaining of this section by describing two new instances of SDYNA, namely ULP and UNATLP.

3.1 Learning the Structure

To be able to learn additively decomposed reward functions, we assume that the reward received by the agent from the environment is not a single real number $r \in \mathbb{R}$ but a vector $r = (r_1, \ldots, r_r) \in \mathbb{R}^r$ where each r_j is the reward associated to the localized R_j function. Consequently, we assume neither the knowledge of the scope nor the structure of context independencies of the localized functions. Figure 2 shows the Learn($\mathcal{F}, \langle s, a, s', r \rangle$) algorithm for both ULP and UNATLP, adapted from SPITI, to learn additively decomposed reward functions.

The transition function in ULP is updated in the same way as the transition function in SPITI (step 1). However, because the reward r observed by the agent is now decomposed as a vector of reward $r = (r_1, \ldots, r_r)$, a different tree $\mathsf{Tree}[R_j]$ corresponding to a localized function R_j in $R(s) = \sum_{j=1}^{r} R_j(s)$ is updated using the current state as input and the reward r_j in r as output (step 2).

Input: a FMDP $\mathcal{F}$, an observation $\langle s, a, s', r \rangle$ Output: $\emptyset$

1. For all $X_i \in X$:
   In ULP: UpdateTree($\mathsf{Tree}[P^a_{X_i}], \langle (x_1, \ldots, x_n), x'_i \rangle$)
   In UNATLP: UpdateTree($\mathsf{Tree}[P_{X_i}], \langle (x_1, \ldots, x_n), a, x'_i \rangle$)
2. For all $R_j \in R$:
   UpdateTree($\mathsf{Tree}[R_j], \langle (x_1, \ldots, x_n), a, r_j \rangle$)

Fig. 2. The Learn($\mathcal{F}, \langle s, a, s', r \rangle$) algorithm in ULP and UNATLP

Both SPITI and ULP use one tree $\mathsf{Tree}[P^a_{X_i}]$ for each variable and each action to represent the CPD $P^a_{X_i}$ in a FMDP $\mathcal{F}$. It is also possible to represent the transition function with the action as a variable [12] with only one CPD P_{X_i} for each variable X_i of the problem to quantify only one graph G (with an action node) specifying variable dependencies in the transition function.

One drawback of this representation is that it is not possible to specify a dependency between two variables for only one action in the graph G. For instance, it is not possible to specify that X'_i depends on X_j for the action a_k, but not for the other actions of the MDP. However, this drawback may be counterbalanced by using structured representation of the CPDs to exploit context specific independence because probability distributions that do not depend on the action executed by the agent can be aggregated.

UNATLP uses such representation of the transition function. The difference between ULP and UNATLP to learn the FMDP is in step 1 (Figure 2) where each CPD $\mathsf{Tree}[P_{X_i}]$, that is the tree representing P_{X_i} with the action considered as a variable, is updated for each variable X_i using the current state and the action executed by the agent as the set of attributes and the value of the variable X_i in s' as input.

3.2 Acting and Planning

SDYNA does not need an explicit representation of the policy for the agent to act. However, it requires a set of action-value functions to select an action with the best action-value. We note $\mathsf{Rules}[F]$ a function F represented as a rule-based function. Recalling equation 1 and using the factored linear value function $\mathsf{Rules}[\mathcal{V}]$, we can obtain the best action by computing $\text{Greedy}_{\mathcal{V}}(s) = \arg\max_a [\sum_{j=1}^{r} \mathsf{Rules}[r^a_j](s) + \gamma \sum_{i=1}^{k} w_i \mathsf{Rules}[g^a_i](s)]$. Both ULP and UNATLP instances use $\text{Greedy}_{\mathcal{V}}(s)$ combined with ϵ-*greedy* as exploration policy, which executes the best action most of the time, and, with a small probability ϵ, selects uniformly at random an action. When different actions are considered as best, one of them is selected uniformly at random.

The LP based method, as described in section 2.1, uses rule-based representations to exploit context-specific structure whereas both ULP and UNATLP use tree representations of CPDs. Nevertheless, from $\mathsf{Tree}[P](X'|s)$, we can build the corresponding rule-based representation $\mathsf{Rules}[P](X'|s)$ by composing the set of probability rules such that: $\mathsf{Rules}[P](X'|s) = \{|c_i \wedge X' = x : p_i| \text{ such that } x \in \text{Dom}(X), p_i(X' = x|s) \neq 0, \forall l_i \in \mathsf{Tree}[P](X'|s)\}$ with c_i the context of the leaf l_i and p_i the probability $P(X' = x|s)$ in l_i. A similar conversion is used to obtain rule-based value functions from decision trees.

By using such conversions, the FMDP incrementally learned may be used with linear programming based planning method. We propose, in Figure 3, an incremental planning algorithm able to exploit the additive structure of a problem for both ULP and UNATLP.

The main idea is to re-use the previous solution of the LP when it is available and to avoid to solve the full LP at each time step. First, the $\text{Plan}(\mathcal{F}_t, \mathsf{Rules}[\mathcal{V}_{t-1}])$ algorithm checks if the structure of the FMDP has changed (such information is

Input: $\mathcal{F}_t, \mathsf{Rules}[\mathcal{V}_{t-1}]$ Output: $\mathsf{Rules}[\mathcal{V}_t], \{\mathsf{Rules}[Q_a^{\mathcal{V}_t}], \forall a \in A\}$, Parameters: T_P, T_M, T_{MIN}

1. If (Structure of $\mathcal{F}_t \neq$ Structure of $\mathcal{F}_{t-1}$) then:
   (a) lastModif $\leftarrow t$
   (b) Reinitialize the solution of the last linear program
2. If $((t - \text{lastModif} > T_M)$ or $(t - \text{lastPlanning} < T_P))$ and $(t - \text{lastPlanning} > T_{MIN})$ then:
   (a) lastPlanning $\leftarrow t$
   (b) $\{\mathsf{Rules}[\mathcal{V}_t], \{\mathsf{Rules}[Q_a^{\mathcal{V}_t}], \forall a \in A\}\} \leftarrow$ FactoredALP($\mathcal{F}_t$) using the solution of the last linear program if it has not been reinitialized.
   else: $\{\mathsf{Rules}[\mathcal{V}_t], \{\mathsf{Rules}[Q_a^{\mathcal{V}_t}], \forall a \in A\}\} \leftarrow \{\mathsf{Rules}[\mathcal{V}_{t-1}], \{\mathsf{Rules}[Q_a^{\mathcal{V}_{t-1}}], \forall a \in A\}\}$
3. Return $\mathsf{Rules}[\mathcal{V}_t]$ and $\{\mathsf{Rules}[Q_a^{\mathcal{V}_t}], \forall a \in A\}$

Fig. 3. The Plan($\mathcal{F}_t, \mathsf{Rules}[\mathcal{V}_{t-1}]$) algorithm in ULP and UNATLP instances of SDYNA

maintained by the UpdateTree algorithm for each tree of the FMDP). When the structure does not change, then the constraints of the LP have the same structure using the same free variables. Consequently, we use the solution of the last linear program as a feasible solution (step 2b).

When the structure has changed, then we cannot re-use the solution of the last linear program which is reinitialized (step 1). Then, during step 2, if the structure has not changed for T_M time step, or if the last time a linear program has been solved is under T_P and over T_{MIN}, then the current solution is updated. When the solution of the linear program has not been updated, then the algorithm returns the last computed solution.

Note that to use the representation of the transition function of UNATLP, an additional step is required: it is necessary to build the CPD $\mathsf{Tree}[P^a_{X_i}]$ for each action a from the CPD $\mathsf{Tree}[P_{X_i}]$. If the action is not tested in $\mathsf{Tree}[P_{X_i}]$, then $\mathsf{Tree}[P^a_{X_i}] = \mathsf{Tree}[P_{X_i}]$ for all a, else $\mathsf{Tree}[P^a_{X_i}]$ is extracted from $\mathsf{Tree}[P_{X_i}]$ by replacing each decision node testing the action in $\mathsf{Tree}[P_{X_i}]$ by the sub-tree corresponding to a.

4 Results

We now present an empirical evaluation of both ULP and UNATLP. We use the SysAdmin benchmark problem, strictly following the specification given in [5] using the unidirectional ring network architecture with $N = 40$ machines. This problem exhibits a very symmetric model and has been used by [5] to show the performance of FactoredALP. But, unlike the results presented below, the structure of the model was assumed to be fully known. Moreover, [3,4,5] show that SPITI is unsuitable for this problem because of the decision trees used to represent the reward and value functions.

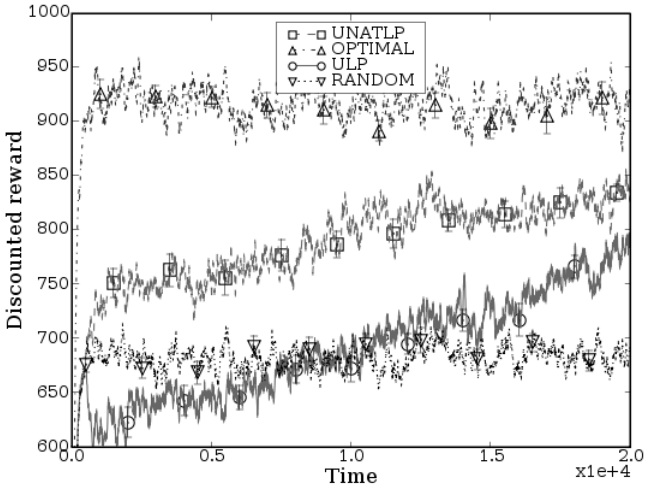

Fig. 4. Discounted reward obtained on the SysAdmin problem. Both ULP and UNATLP are able to improve their policy, despite the large size of the problem. Error bars are present, but hardly visible because of a very low standard deviation.

The size of the problem is $2^{40} \times 41 \approx 4 \cdot 10^{13}$ state/action pairs. A each time step, a SysAdmin may go to one machine to reboot it, making it work for the next time step with a high probability. The SysAdmin receives a reward of 1 for each running machine (except for one machine for which it receives 2 to introduce an asymmetry in the problem). We use N basis functions h_i corresponding to each machine represented by the X_i variable and defined such as $\{|X_{i-1} = 0 \wedge X_i = 0 : 0.05|, |X_{i-1} = 1 \wedge X_i = 0 : 0.09|, |X_{i-1} = 0 \wedge X_i = 1 : 0.5|, |X_{i-1} = 1 \wedge X_i = 1 : 0.9|\}$ and a constant basis function.

We use `glpsol`[1] as LP solver, $\epsilon = 0.1$ in ϵ-*greedy* exploration policy, a discount factor $\gamma = 0.99$, $\tau_{\chi^2} = 30$ in the UpdateTree induction tree algorithm and $T_M = 100$, $T_P = 1500$ and $T_{MIN} = 50$ in the Plan algorithm (Figure 3). We ran 10 experiments of 20,000 time steps. We use the tree induction algorithm ID4 [13]. For implementation reasons, the backprojections $g_i^a(s)$ are computed using decision trees as representations [3]. We refer to the respective papers for respective descriptions. Moreover, we use the least-square criterion [14] to learn reward functions in the UpdateTree algorithm. We ran two more agents, noted RANDOM and OPTIMAL, executing at each time step, respectively, a random action and the best action. The policy of OPTIMAL has been computed off-line, using FactoredALP with the same basis functions as defined below.

Figure 4 shows the discounted reward, defined as $R_t^{disc} = r_t + \gamma R_{t-1}^{disc}$ with r_t the reward received by the agent, obtained by each agent over the time (in time steps). Both ULP and UNATLP are able to substantially improve their policy, compared to RANDOM. Moreover, UNATLP improves fairly quickly its policy compared to ULP and considering the size and the stochasticity of the problem.

[1] http://www.gnu.org/software/glpk/glpk.html

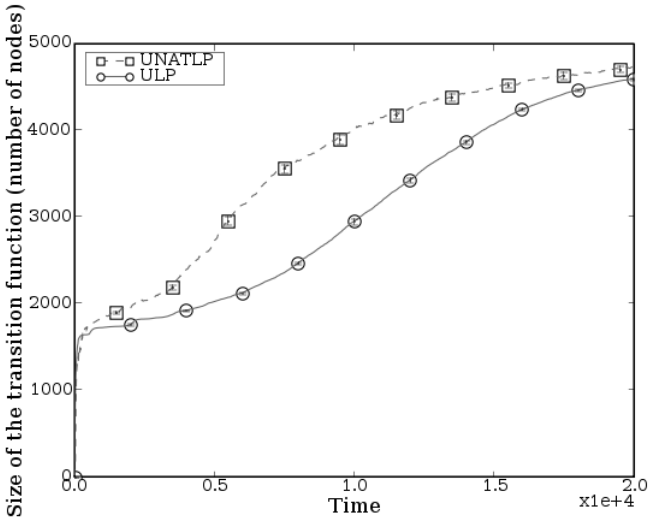

Fig. 5. Size of the transition function of the FMDP learned in ULP and UNATLP. Erros bars are present, but hardly visible because of a very low standard deviation.

Figure 5 compares the different size of the transition function (in nodes, including decision nodes and leaves) learned by ULP and UNATLP. Both build a transition function with a similar size of approximately 4500 nodes (compared to the size of $41 \times 40 \times 7 = 11480$ nodes of the full transition function with perfect knowledge using the ULP representation). We observe that, for a similar size, UNATLP obtains better results than ULP. Moreover, the transition function built by UNATLP grows quicker than the transition function in ULP.

5 Discussion

These results show that ULP and UNATLP are able to quickly learn an accurate FMDP representing the RL problem to solve with few assumptions on this problem. They are able to exploit the additive structure of a problem even when its structure is not known in advance. Thus, by combining LP based planning methods with supervised learning methods such as decision tree induction, ULP and UNATLP are able to address very large problems by exploiting a strong generalisation property. Note however that the size of the model stabilizes in both cases below the size of the perfect model, which suggests that the perfect representation will not be learned. This is mainly due to the ϵ-*greedy* exploration policy we use, which drives the agent to explore only paths near the greedy policy. However, despite an imperfect representation, these results clearly show that the policy is still improved. Including better exploration policy than ϵ-*greedy* is still work in progress.

Moreover, these results illustrate the difference between representations of the transition function in ULP and UNATLP. Figure 4 shows that UNATLP learns

quicker than ULP. The main reason is that new examples only update the CPD of the last action executed in ULP, whereas they update all the trees of the transition function in UNATLP. [12] suggests that the second representation may be more compact when the value of a variable persists for all actions. However, such a property could not be observed in this problem because each variable depends on one action. Despite the fact that the ULP representation is usually used in FMDP planning algorithms, these results suggest that the representation used in UNATLP is an interesting alternative to solve RL problems.

Future works include a method for building automatically a good set of basis functions, such as the work of [15]. To our knowledge, ULP and UNATLP are the first algorithms to learn and to exploit a FMDP with a factored transition function and an additively decomposed reward function. Such FMDP can directly be used to construct a set of localized basis functions adapted to FMDP planning methods. Thus, such approach would be very promising, combining the generality of the representations used in ULP and UNATLP with the automation of SPITI to solve large RL problems.

Our first contribution in this paper was to show that LP planning methods may be combined with decision tree induction to address very large RL problems, exploiting additive structure, even when the structure is unknown. Our second contribution is to show that a different representation of the transition function in FMDPs may speed up the learning process, particularly in large RL problems, with no loss on the size of the transition function. To conclude, approaches such as ULP and UNATLP can address RL problems that were out of reach with previous model based methods, as far as no information at all about the structure of the problem is given to the system.

References

1. Boutilier, C., Dearden, R., Goldszmidt, M.: Exploiting Structure in Policy Construction. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence, pp. 1104–1111 (1995)
2. Dean, T., Kanazawa, K.: A Model for Reasoning about Persistence and Causation. Computational Intelligence 5, 142–150 (1989)
3. Boutilier, C., Dearden, R., Goldszmidt, M.: Stochastic Dynamic Programming with Factored Representations. Artificial Intelligence 121(1), 49–107 (2000)
4. Hoey, J., St-Aubin, R., Hu, A., Boutilier, C.: SPUDD: Stochastic Planning using Decision Diagrams. In: Proceedings of the 15th Conference on UAI, pp. 279–288. Morgan Kaufmann, San Francisco (1999)
5. Guestrin, C., Koller, D., Parr, R., Venkataraman, S.: Efficient Solution Algorithms for Factored MDPs. Journal of Artificial Intelligence Research 19, 399–468 (2003)
6. Degris, T., Sigaud, O., Wuillemin, P.H.: Learning the Structure of Factored Markov Decision Processes in Reinforcement Learning Problems. In: Proceedings of the 23rd ICML, pp. 257–264 (2006)
7. Manne, A.S.: Linear Programming and Sequential Decisions. Cowles Foundation for Research in Economics at Yale University (1960)
8. Schweitzer, P., Seidmann, A.: Generalized Polynomial Approximations in Markovian Decision Processes. Journal of Mathematical Analysis and Applications 110, 568–582 (1985)

9. Koller, D., Parr, R.: Computing Factored Value Functions for Policies in Structured MDPs. In: Proceedings 16th International Joint Conference on Artificial Intelligence, pp. 1332–1339 (1999)
10. Zhang, T., Poole, D.: On the Role of Context-specific Independence in Probabilistic Reasoning. In: Proceedings of the 16th International Joint Conference on Artificial Intelligence, pp. 1288–1293 (1999)
11. Degris, T., Sigaud, O., Wuillemin, P.H.: Chi-square Tests Driven Method for Learning the Structure of Factored MDPs. In: Proceedings of the 22nd Conference on UAI, pp. 122–129 (2006)
12. Boutilier, C., Goldszmidt, M.: The Frame Problem and Bayesian Network Action Representations. In: Proceedings of the Eleventh Biennial Canadian Conference on Artificial Intelligence, pp. 69–83 (1996)
13. Schlimmer, J., Fisher, D.: A Case Study of Incremental Concept Induction. In: Proceedings of the Fifth National Conference on Artificial Intelligence, pp. 496–501 (1986)
14. Breiman, B., Breiman, L.: Classification and Regression Trees. Chapman & Hall/CRC, Boca Raton (1984)
15. Kveton, B., Hauskrecht, M.: Learning Basis Functions in Hybrid Domains. In: Proceedings of the 21st National Conference on Artificial Intelligence, pp. 1161–1166 (2006)