Efficient Structure Learning in Factored-state MDPs

Alexander L. Strehl, Carlos Diuk and Michael L. Littman
RL^3 Laboratory
Department of Computer Science
Rutgers University
Piscataway, NJ USA
{strehl,cdiuk,mlittman}@cs.rutgers.edu

Abstract

We consider the problem of reinforcement learning in factored-state MDPs in the setting in which learning is conducted in one long trial with no resets allowed. We show how to extend existing efficient algorithms that learn the conditional probability tables of dynamic Bayesian networks (DBNs) given their structure to the case in which DBN structure is not known in advance. Our method learns the DBN structures as part of the reinforcement-learning process and provably provides an efficient learning algorithm when combined with factored Rmax.

Introduction

In the standard Markov Decision Process (MDP) formalization of the *reinforcement-learning* (RL) problem (Sutton & Barto 1998), a decision maker interacts with an environment consisting of finite state and action spaces. Learning in environments with extremely large state spaces is challenging if not infeasible without some form of generalization. Exploiting the underlying structure of a problem can effect generalization and has long been recognized as an important aspect in representing sequential decision tasks (Boutilier, Dean, & Hanks 1999).

A factored-state MDP is one whose states are represented as a vector of distinct components or features. Dynamic Bayesian networks (DBNs) and decision trees are two popular formalisms for succinctly representing the state-transition dynamics of factored-state MDPs, rather than enumerating such dynamics state by state (Guestrin, Patrascu, & Schuurmans 2002). We adopt these powerful formalisms.

Algorithms for provably experience-efficient exploration of MDPs have been generalized to factored-state MDPs specified by DBNs. Factored E^3 (Kearns & Koller 1999) and Factored Rmax (Guestrin, Patrascu, & Schuurmans 2002) are known to behave near optimally, with high probability, in all but a polynomial number of timesteps. Unfortunately, these algorithms require as input a complete and correct DBN structure specification (apart from the CPT parameters), which describes the exact structural dependencies among state variables.

There has been recent interest in learning the underlying structure of a problem from data, especially within the RL framework. Degris, Sigaud, & Wuillemin (2006) introduce a model-based algorithm that incrementally builds a decision-tree representation of state transitions. Their method, while successful in challenging benchmark domains, incorporates a very simplistic exploration technique (ϵ-greedy) that is known to miss important opportunities for exploration. In addition, it has no formal performance guarantees. Abbeel, Koller, & Ng (2006) show that factor graphs, a generalization of DBNs, can be learned in polynomial time and sample complexities. Their method, which is not based on maximum likelihood estimation, assumes access to i.i.d. samples during separate training and testing phases. While our approach is inspired by theirs, we address the RL problem, which requires dealing with highly dependent (non i.i.d.) inputs, temporal considerations, and the exploration-exploitation tradeoff.

Our paper makes two important contributions. First, we integrate structure learning with focused exploration into a complete RL algorithm. Second, we prove that our algorithm uses its experience efficiently enough to provide formal guarantees on the quality of its online behavior.

We deal almost solely with the problem of maximizing experience, rather than computational, efficiency. Using supervised learning terminology as a metaphor, we seek to minimize sample rather than computational complexity. Our algorithm relies on access to an MDP planner (value iteration in our experiments), which is often very costly in terms of computation. For a more practical implementation, faster approximate planners could be employed. (For examples, see the paper by Degris, Sigaud, & Wuillemin, 2006.)

In the next section, we discuss and formally describe MDPs and factored-state MDPs. Next, we introduce two common structures—dynamic Bayesian networks and decision trees—for succinct representation of the transition dynamics in factored-state MDPs. We formulate the general "Structure-Learning Problem" as an online learning-theory problem that involves prediction of the probability an output of 1 will be observed for a specified input. We provide the "Basic Structure-Learning Algorithm" as a concrete solution to a simple instance of this problem. We prove that, with high probability, the number of errors (refusals to predict) made by the algorithm is small (polynomial in the size

of the problem representation). In the following section, we return to the reinforcement-learning problem and introduce our new algorithm, SLF-Rmax. The algorithm uses the solution to the online learning-theory problem just described. We prove that the SLF-Rmax algorithm acts according to a near-optimal policy on all but a small number of timesteps, with high probability. This bound is inherently linked, through reduction, with the corresponding error bound of the online learning algorithm. We briefly discuss how the Basic Structure-Learning Algorithm can be extended and incorporated into SLF-Rmax to efficiently solve the general structure-learning RL problem using either the DBN or decision-tree models. Next, we perform an empirical comparison of SLF-Rmax with Rmax and Factored Rmax. These algorithms differ according to how much prior knowledge they have—Rmax neither uses nor requires underlying structure; SLF-Rmax is told there is structure but must find it itself, and Factored Rmax requires complete knowledge of the underlying structure. The performance of the algorithms reflects the amount of knowledge available in that the more background provided to the algorithm, the faster it learns.

Background

This section introduces the Markov Decision Process (MDP) notation used throughout the paper (Sutton & Barto 1998). Let $\mathcal{P}_S$ denote the set of all probability distributions over the set S.

Definition 1 *A* **finite MDP** *M is a five tuple $\langle \mathcal{S}, \mathsf{A}, T, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ is a finite set called the state space, A is a finite set called the action space, $T : \mathcal{S} \times \mathsf{A} \to \mathcal{P}_{\mathcal{S}}$ is the transition function, $\mathcal{R} : \mathcal{S} \times \mathsf{A} \to \mathcal{P}_{\mathbb{R}}$ is the reward function, and $0 \leq \gamma < 1$ is a discount factor on the summed sequence of rewards.*

We call the elements of $\mathcal{S}$ and A states and actions, respectively, and define $S = |\mathcal{S}|$ and $A = |\mathsf{A}|$. We use $T(s'|s,a)$ to denote the transition probability of state s' in the distribution $T(s,a)$ and $R(s,a)$ to denote the expected value of the distribution $\mathcal{R}(s,a)$.

A *policy* is any strategy for choosing actions. A stationary policy is one that produces an action based on only the current state, ignoring the rest of the agent's history. We assume, without loss of generality, that rewards all lie in the interval $[0,1]$. For any policy π, let $V_M^\pi(s)$ ($Q_M^\pi(s,a)$) denote the discounted, infinite-horizon value (action-value) function for π in M (which may be omitted from the notation) from state s. Specifically, for any state s and policy π, let r_t denote the tth reward received after following π in M starting from state s. Then, $V_M^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t | s, \pi]$. The optimal policy is denoted π^* and has value functions $V_M^*(s)$ and $Q_M^*(s,a)$. Note that a policy cannot have a value greater than $1/(1-\gamma)$.

Factored-state MDPs

Definition 2 *A* **factored-state MDP** *is an MDP where the states are represented as vectors of n components $X = \{X_1, X_2, \ldots, X_n\}$. Each component X_i (called a* **state variable** *or* **state factor***) may be one of finitely many values from the set $\mathcal{D}(X_i)$. In other words, each state can be written in the form $x = \langle x(1), \ldots, x(n) \rangle$, where $x(i) \in \mathcal{D}(X_i)$.*

The goal of factored representations is to succinctly represent large state spaces. The number of states of a factored-state MDP M is *exponential* in the number n of state variables. To simplify the presentation, we assume the reward function is known and does not need to be learned. We also assume that each factor is binary valued ($\mathcal{D}(X_i) = \{0,1\}$). All of our results have straightforward extensions to the case of an unknown reward function and multi-valued factors.

Now, we make a mild independence assumption (that can be relaxed in some settings).

Assumption 1 *Let s, s' be two states of a factored-state MDP M, and a an action. The transition distribution function satisfies the following conditional independence condition:*

$$T(s'|s,a) = \prod_i P_i(s'(i)|s,a), \tag{1}$$

where $P_i(\cdot|s,a)$ is a discrete probability distribution over $\mathcal{D}(X_i)$ for each factor X_i and state-action pair (s,a). Said another way, the DBNs have no synchronic arcs.

This assumption ensures that the values of each state variable after a transition are determined independently of each other, conditioned on the previous state and action.

The learning algorithms we consider are allowed to interact with the environment only through one long trajectory of (state, action, reward, next-state) tuples, governed by the system dynamics above. The transition function is not provided to the algorithm and must be learned from scratch.

Different Models

Factored-state MDPs are mainly useful when there are restrictions on the transition function that allows it to be represented by a data structure with reduced size. The corresponding goal of a learning algorithm is to achieve a learning rate that is polynomial in the representation size. Two different representations, DBNs and decision trees, are discussed in this section. As an example of the different expressive powers of these representations, we refer the reader to the taxi domain (Dietterich 2000), a grid world in which a taxi has to pick up a passenger from one of four designated locations and drop it off at a different one. The state space can be factored into 3 state variables: taxi position, passenger location and passenger destination.

The dynamic Bayesian network (DBN) framework is one model commonly used to describe the structure of factored-state MDPs (Boutilier, Dean, & Hanks 1999). This model restricts the set of factors that may influence the value of a specified factor after a specified action. For example, it is powerful enough to capture the following relationship in the taxi domain: the factor that indicates the passenger location depends only on itself (and not, say, the destination) after a "move forward" command. Several learning methods have been developed for factored-state MDPs that require the DBNs structures themselves (but not the CPTs) as input (Kearns & Koller 1999; Guestrin, Patrascu, & Schuurmans 2002). Our new algorithm operates when provided

only an upper bound on the max degree of the DBNs (equivalently, an upper bound on the number of parents of any factor).

Although DBNs are quite useful, they fail to succinctly represent certain dependencies. For example, in the taxi domain, the value of the passenger variable after a "drop off" command is unchanged unless the taxi contains the passenger and is in the destination. The DBN representation for this dependency would indicate that the passenger variable depends on all three state variables (position, passenger, destination). On the other hand, a simple decision tree with two internal nodes can test whether the passenger is in the taxi and whether the taxi is in the destination. The decision-tree representation is an order of magnitude smaller than the DBN representation that uses tabular CPTs, in this case. Here, we allow the nodes of the decision trees to be simple decision rules of the form $x(i) = z$, where $i \in \{1, \ldots, n\}$ is a factor and $z \in \{0, 1\}$ is a literal. Our algorithm has the ability to learn in factored-state MDPs whose transition functions are specified by decision trees. It needs only be given a bound on the depth of the trees.

Structure-Learning Problem

As we will show, the structure-learning problem for MDPs boils down to the following simple on-line learning-theory problem:

Definition 3 (Structure-Learning Problem) *On every step $t = 1, 2, \ldots$ an input vector $x_t \in \{0,1\}^n$ and output bit $y_t \in \{0,1\}$ is provided. The input x_t may be chosen in any way that depends only on the previous inputs and outputs $(x_1, y_1), \ldots, (x_t, y_t)$. The output y_t is chosen with probability $P(x_t)$, where $P(x_t)$ depends only on the input x_t. After observing x_t and before observing y_t, the learning algorithm must make a prediction $\hat{P}_t(x_t) \in [0,1] \cup \{\emptyset\}$ of $P(x_t)$. Furthermore, it should be able to provide a prediction $\hat{P}_t(x)$ for any input vector $x \in \{0,1\}^n$.*

We require that $\hat{P}_t(x)$ is a very accurate prediction of the probability, $P(x)$, that $y = 1$ given input x. If the algorithm cannot make an accurate prediction, it must choose $\emptyset$.

Definition 4 *We define an* **admissible algorithm for the Structure-Learning Problem** *to be one that takes two inputs $0 \le \epsilon \le 1$ and $0 \le \delta < 1$ and, with probability at least $1 - \delta$, satisfies the following conditions:*

1. *Whenever the algorithm predicts $\hat{P}_t(x) \in [0,1]$, we have that $|\hat{P}_t(x) - P(x)| \le \epsilon$.*
2. *If $\hat{P}_t(x) \neq \emptyset$ then $\hat{P}_{t'}(x) \neq \emptyset$ for all $t' > t$.*
3. *The number of timesteps t for which $\hat{P}_t(x_t) = \emptyset$ is bounded by some function $\zeta(\epsilon, \delta)$, polynomial in ϵ and δ.*

The first condition above requires the algorithm to predict accurately or refrain from predicting (by choosing $\emptyset$). The second condition states that once a valid ($\neq \emptyset$) prediction is made for input x, then valid predictions must be provided for x in the future. This condition can easily be met by simply remembering all valid predictions. The function $\zeta(\epsilon, \delta)$ in the third condition above is called the **learning complexity** of the algorithm and represents the number of *acknowledged mistakes* (predictions of $\emptyset$ for a given input x_t) made by the algorithm.

As stated, the Structure-Learning problem is solvable only by exhaustive input enumeration (wait for each of the 2^n input bit vectors to be seen a sufficient number of times). However, if additional assumptions on the probability distributions P_t are made, then faster learning through generalization is possible. As one example of a problem that allows generalization, suppose that the output probability P_t depends only on the i^*th bit of the input x_t, where the identity of the specially designated bit i^* is not provided to the algorithm. We call this scenario the **Basic Structure-Learning Problem**. Later, we will discuss how our solution to this problem can be generalized to handle more than a single designated bit and to deal with the various modeling assumptions discussed in the previous section.

Basic Structure-Learning Algorithm

Our solution to the Basic Structure-Learning Problem is specifically designed to be the simplest algorithm that easily generalizes to more realistic models. Intuitively, given a new input x, we must predict $P(x)$, which depends only on $x(i^*)$, the i^*th component of x. If the algorithm knew which bit, i^*, mattered, it could easily estimate $P(x)$ from a few sample input/output pairs whose inputs agree with x on bit i^*. Since i^* is initially unknown, our algorithm keeps empirical counts on the number of times an observed output bit of 1 occurs given an input whose ith and jth components are fixed at some setting. It keeps these statistics for all pairs of bit positions (i, j) and valid settings to these two bit positions. This method is based on the observation that the correct bit position i^* satisfies $P(\cdot|x(i^*) = z_1, x(j) = z_2) = P(\cdot|x(i^*) = z_1, x(j') = z_2')$ for all other bit positions j and j' and bits z_1, z_2, z_2'. For a given input x, the algorithm searches for a bit position that approximately satisfies this relationship. If one is found, a valid prediction is computed based on past observations, and, if not, the algorithm predicts $\emptyset$.

Formally, our algorithm works as follows. It requires the following parameters as input:

- *Experience threshold* $m \in \mathbb{Z}^+$.
- *Precision parameter* $\epsilon_1 \in \mathbb{R}^+$.

The parameters essentially quantify the algorithm's required level of accuracy. The algorithm maintains the following local variables:

- *Position-pair counts.* For every pair of distinct bit positions $(i, j) \in \{1, \ldots, n\}^2$ with $i \neq j$ and every pair of bits $z = (z_1, z_2) \in \{0,1\}^2$, the quantity $C(i, j, z)$ is the minimum of m and the number of timesteps t the algorithm has experienced an input-output pair (x_t, y_t) with $x_t(i) = z_1$ and $x_t(j) = z_2$.
- *Next-bit counts.* For every pair of distinct bit positions $(i, j) \in \{1, \ldots, n\}^2$ and every pair of bits $z = (z_1, z_2) \in \{0,1\}^2$, the quantity $C(1|i, j, z)$ is the number of timesteps t during which the algorithm has experienced

an input-output pair (x_t, y_t) with $x_t(i) = z_1$, $x_t(j) = z_2$, $y_t = 1$, and $C_t(i, j, z) < m$.

During timestep t, the algorithm is provided an input x_t and must make a prediction $\hat{P}_t(x_t)$. It must also be able to produce a prediction $\hat{P}_t(x)$ for any bit vector x. For each queried bit vector x, our algorithm searches for a bit position i such that the following conditions hold:

- For all bit positions $j \neq i$, the algorithm has experienced at least m samples that agree with x in the ith and jth component. Formally, $C(i, j, (x(i), x(j))) = m$ for all $j \neq i$.
- For all bit positions $j \neq i$, the number of times the agent has observed an output of 1 after experiencing an input that matches x in the ith and jth component (considering only the first m such samples) lie within an $m\epsilon_1$ ball. Formally, $|C(1|i, j, (x(i), x(j))) - C(1|i, j', (x(i), x(j')))| \leq m\epsilon_1$, for all j, j'.

If such a bit position i is found, then the algorithm uses

$$\hat{P}_t(x) = \frac{C(1|i, j, (x(i), x(j)))}{C(i, j, (x(i), x(j))} = \frac{C(1|i, j, (x(i), x(j)))}{m}$$

as its prediction (for any $j \neq i$ chosen arbitrarily). Otherwise, the algorithm makes the null prediction $\emptyset$.

Theorem 1 *The inputs $m \propto \frac{\ln(n/\delta)}{\epsilon^2}$ and $\epsilon_1 \propto \epsilon$ can be chosen so that the Basic Structure-Learning Algorithm described in this section is an admissible learning algorithm with $\zeta(\epsilon, \delta) \propto \frac{n^2 \ln(n/\delta)}{\epsilon^2}$.*

Proof sketch: The proof has four steps. First, we consider a fixed setting $(z_1, z_2) \in \{0, 1\}^2$ to a pair of bit positions, (i^*,j) that includes the correct factor. Using Hoeffding's bound, we show that if m independent samples of the next bit y are obtained from inputs x matching this setting ($x(i^*) = z_1, x(j) = z_2$), then it is very unlikely for the algorithm to learn an incorrect prediction of y given this fixed setting (formally $|P(x) - C(1|i^*, j, (z_1, z_2))/m| \leq \epsilon_1$ for all x such that $x(i^*) = z_1$). Second, we observe that even though each input can be chosen in an adversarial manner (dependent only on the past inputs and outputs), the output is always chosen from a fixed distribution (dependent on only the input). Thus, the probability of an incorrect prediction in the adversarial setting is no greater than when m independent samples are available. Third, we use a union bound over the $4n$ different pairs of factors and binary settings to show that all predictions involving a correct factor are accurate, with high probability. The rest of the argument proceeds as follows. If a prediction is made for input x that is not $\emptyset$, then it is equal to $C(1|i, j', (x(i), x(j')))/m$ for some factor i according to the conditions above. Suppose that i is not the correct factor i^*. The second condition (right above the theorem) implies that $|C(1|i, j', (x(i), x(j')))/m - C(1|i, i^*, (x(i), x(i^*)))/m| \leq \epsilon_1$. We have shown that with high probability, $|P(x) - C(1|i, i^*, (x(i), x(i^*)))/m| \leq \epsilon_1$. Combining these two facts gives $|C(1|i, j', (x(i), x(j')))/m - P(x)| \leq 2\epsilon_1$. Thus, we choose $\epsilon_1 = \epsilon/2$ to satisfy condition (1) of Definition 4. Finally, a simple counting argument and an application of the pigeonhole principle yield the bound on $\zeta(\epsilon, \delta)$. □

The General SLF-Rmax Algorithm

In this section, we describe our new structure-learning algorithm called **Structure-Learn-Factored-Rmax** or **SLF-Rmax**. First, we provide the intuition behind the algorithm, then we define it formally. The algorithm is model based. During each timestep, it acts according to a near-optimal policy of its model, which it computes using any MDP planning algorithm. Since the true transition probabilities are unknown, the algorithm must learn them from past experience. Each transition component, $P_i(\cdot|s, a)$, is estimated from experience by an instance of any admissible learning algorithm for the Structure-Learning problem described in the previous section. Thus, SLF-Rmax uses nA instantiations, $\mathcal{A}_{i,a}$, of this algorithm, one for each factor i and action a. When viewed from $\mathcal{A}_{i,a}$'s perspective (as in the last section), each input vector x is a state and each output bit y is the ith bit position of the next state reached after taking action a from x. If any estimated transition component $\hat{P}_i(\cdot|s, a)$ is $\emptyset$, meaning that the algorithm has no reasonable estimate of $P_i(\cdot|s, a)$, then the value of taking action a from state s in SLF-Rmax's model is the maximum possible $(1/(1 - \gamma))$. Similar to the Rmax algorithm (Brafman & Tennenholtz 2002), this *exploration bonus* is an imaginary reward for experiencing a state-action pair whose next-state distribution involves an inaccurately-modeled transition component.

Formally, SLF-Rmax chooses an action from state s that maximizes its current action-value estimates $Q(s, a)$, which are computed by solving the following set of equations:

$$\begin{aligned} Q(s, a) &= 1/(1 - \gamma), \text{if } \exists i,\ \hat{P}_i(\cdot|s, a) = \emptyset \qquad (2) \\ Q(s, a) &= R(s, a) + \gamma \sum_{s'} \hat{T}(s'|s, a) \max_{a'} Q(s', a'), \end{aligned}$$

otherwise.

In Equation 2, we have used $\hat{T}(s'|s, a) = \prod_i \hat{P}_i(s'(i)|s, a)$. Pseudo-code for the SLF-Rmax algorithm is provided in Algorithm 1.

Theoretical Analysis

We can prove that when given an admissible learning algorithm for the Structure-Learning problem, SLF-Rmax behaves near-optimally on all but a few timesteps, with high probability. The result is comparable with the standard polynomial-time guarantees of RL algorithms (Kearns & Singh 2002; Kakade 2003; Brafman & Tennenholtz 2002).

Theorem 2 *Suppose that $0 \leq \epsilon < \frac{1}{1-\gamma}$ and $0 \leq \delta < 1$ are two real numbers and $M = \langle \mathcal{S}, \mathsf{A}, T, \mathcal{R}, \gamma \rangle$ is any factored-state MDP. Let n be the number of state factors. Suppose that $\mathcal{S}tructLearn$ is an admissible learning algorithm for the Basic Structure-Learning Problem that is used by SLF-Rmax and has learning complexity $\zeta(\epsilon, \delta)$. Let $\mathcal{A}_t$ denote*

Algorithm 1 SLF-Rmax Algorithm

Inputs: n, A, R, γ, ϵ, δ, admissible learning algorithm $\mathcal{S}tructLearn$
for all factors $i \in \{1, \ldots, n\}$ and actions $a \in \mathsf{A}$ **do**
  Initialize a new instantiation of $\mathcal{S}tructLearn$, denoted $\mathcal{A}_{i,a}$, with inputs $\epsilon(1-\gamma)^2/n$, and $\frac{\delta}{nk}$, respectively (for ϵ and δ in Definition 4).
end for
for all $(s, a) \in \mathcal{S} \times \mathsf{A}$ **do**
  $Q(s, a) \leftarrow 1/(1-\gamma)$ // *Action-value estimates*
end for
for $t = 1, 2, 3, \cdots$ **do**
  Let s denote the state at time t.
  Choose action $a := \text{argmax}_{a' \in A}\, Q(s, a')$.
  Let s' be the next state after executing action a.
  for all factors $i \in \{1, \ldots, n\}$ **do**
    Present input-output pair $(s, s'(i))$ to $\mathcal{A}_{i,a}$.
  end for
  Update action-value estimates by solving Equation 2.
end for

SLF-Rmax's policy at time t and s_t denote the state at time t. With probability at least $1-\delta$, $V_M^{\mathcal{A}_t}(s_t) \geq V_M^*(s_t) - \epsilon$ is true for all but

$$O\left(\frac{nA}{\epsilon(1-\gamma)^2}\zeta\left(\frac{\epsilon(1-\gamma)^2}{n}, \frac{\delta}{nA}\right)\ln\frac{1}{\delta}\ln\frac{1}{\epsilon(1-\gamma)}\right)$$

timesteps t.

SLF-Rmax With Different Modeling Assumptions

The general SLF-Rmax algorithm requires, as input, an admissible algorithm for the Structure-Learning problem. For different structural assumptions, different admissible algorithms can be formulated. In a previous section, we provided an algorithm, the Basic Structure-Learning Algorithm, which is admissible under the structural assumption that the probability that the next output bit is 1 depends on a single input bit. This algorithm can be directly incorporated into the SLF-Rmax algorithm for use in factored-state domains whose transition functions are described by decision trees with depth 1. Similarly, it could be used in factored-state domains whose transition functions are described by DBNs with maximum degree 1 (one parent per next-state factor).

In most realistic domains described by decision tress or DBNs, the maximum depth or maximum degree, respectively, will be larger than one but often much smaller than n (the number of factors). The Basic Structure-Learning algorithm can be extended to accommodate this situation. The extended version requires a bound, k, on either the maximum depth of the decision trees or the maximum degree of the DBNs. The main idea of the extension is to note that the true probability associated with the next bit is dependent on some unknown *element* (for instance, a decision-tree leaf or some setting to the parents in a DBN). The algorithm enumerates all possible elements and keeps statistics (empirical probability that the next bit is 1) for all possible pairs of elements. In the case of the Basic Structure-Learning algorithm, the elements are the $2n$ settings to each of the n bits. More generally, there are $2^k\binom{n}{k}$ elements correspond to sets of k input positions and binary strings (settings) over them.

Using the extension of the Basic Structure-Learning algorithm in conjunction with SLF-Rmax, we have the following corollary to Theorem 2. It says that the number of times the algorithm fails to behave near-optimally is bounded by a polynomial in the representation size **when k is a fixed constant**. This exponential dependence on k is unavoidable and appears in similar theoretical results for structure learning (Abbeel, Koller, & Ng 2006).

Corollary 1 *Suppose that $0 \leq \epsilon < \frac{1}{1-\gamma}$ and $0 \leq \delta < 1$ are two real numbers and $M = \langle \mathcal{S}, \mathsf{A}, T, \mathcal{R}, \gamma \rangle$ is any factored-state MDP whose transition function is described by depth-k decision trees (or DBNs with maximum degree k). Let n be the number of state factors. There exists an admissible learning algorithm $\mathcal{S}tructLearn$ so that if SLF-Rmax is executed on M using $\mathcal{S}tructLearn$, then the following holds. Let $\mathcal{A}_t$ denote SLF-Rmax's policy at time t and s_t denote the state at time t. With probability at least $1-\delta$, $V_M^{\mathcal{A}_t}(s_t) \geq V_M^*(s_t) - \epsilon$ is true for all but*

$$O\left(\frac{n^{3+2k}Ak\ln(nA/\delta)\ln\frac{1}{\delta}\ln\frac{1}{\epsilon(1-\gamma)}}{\epsilon^3(1-\gamma)^6}\right)$$

timesteps t.

Proof sketch: The proof of Theorem 2 utilizes an existing theoretical framework (Strehl, Li, & Littman 2006) and is a straightforward extension of the proof for the corresponding result about Rmax (Kakade 2003). Although there is not enough room here, please see our technical report for full details. □

Experiments

In this section, we present a small-scale empirical evaluation that provides a proof of concept and demonstrates how our approach can exploit weak background knowledge. We compare the cumulative reward of three RL algorithms over time. Each is designed to accept different kinds of input: Factored Rmax requires the entire DBN structure; SLF-Rmax requires only an upper bound on the degree of the underlying DBN; and Rmax uses none of the available structural information. The three algorithms are demonstrated on a simplified Stock-Trading domain.

Stock-Trading domain The domain simulates a stock market composed of a set of *economy sectors*, each associated with a set of stocks. The size of the domain is defined by the number of sectors (e) and the number of stocks per sector (o). The domain consists of two types of binary variables: *e sector ownership variables* representing whether or not the agent owns a sector, and *$e \times o$ stock variables*, representing whether each of the individual stocks is rising or falling. The probability that any given stock will be rising at time $t+1$ is determined by a combination of the values of all stocks in its sector at time t, according to the formula $P(stock\ rising) = 0.1 + 0.8 \times (\#stocks\ in\ sector\ rising\ at\ time\ t / \#stocks\ in\ sector)$. The dynamics of these transitions can thus be modeled by

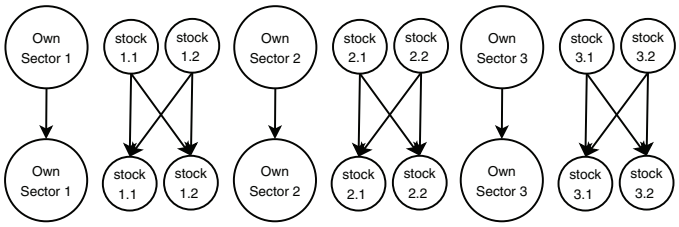

Figure 1: Dynamic Bayesian network representation of the transition dynamics of a 3×2 Stock-Trading environment. The large nodes represent stock ownership variables and the smaller nodes represent stock variables.

a DBN with at most o parents per state variable. Figure 1 shows a DBN structure for a sample ($e \times o = 3 \times 2$) domain.

The agent gets a reward of $+1$ for each stock that is rising in a sector that it owns, and -1 for each stock that is not rising. For stocks in sectors that the agent does not own, the reward is 0 regardless of whether they are rising or dropping. The maximum possible reward in a timestep would thus be $e \times o$, which occurs when the agent owns all sectors and all stocks are rising. The agent's actions are to buy/sell sectors or simply do nothing. To summarize, in the Stock-Trading domain with $e = 3$ and $o = 2$, there are $n = 9$ factors, $S = 2^9 = 512$ states, and $A = 4$ actions. We executed Factored Rmax and SLF-Rmax for 4000 steps, and Rmax, which required more steps to converge, for 14000 steps in the domain. A parameter search over a coarse grid was conducted for each algorithm and the best setting was selected ($m = 10$ for Factored Rmax, $m = 20$ for the other two algorithms; $\epsilon_1 = 0.2$ in all cases). Each experiment was repeated 20 times and the results averaged.

Results Figure 2 shows the reward accumulated by each agent per step of experience. As expected, the fastest algorithm to converge to a near-optimal policy and maximize reward is Factored Rmax, which uses the most prior knowledge. Our new algorithm, SLF-Rmax, converges to a near-optimal policy after only a small number of additional steps, presumably needed to infer the underlying structure. Rmax, which uses the least amount of prior knowledge, eventually found an adequate policy but only after many steps.

Conclusion

SLF-Rmax is the first provably experience-efficient RL algorithm to automatically learn and exploit the structure in factored-state MDPs. Future work includes making the algorithm more practical, by integrating it with approximate planning methods that would enable the solution of larger problems.

Acknowledgments

This material is based upon work supported by NSF ITR 0325281, IIS-0329153, and DARPA HR0011-04-1-0050.

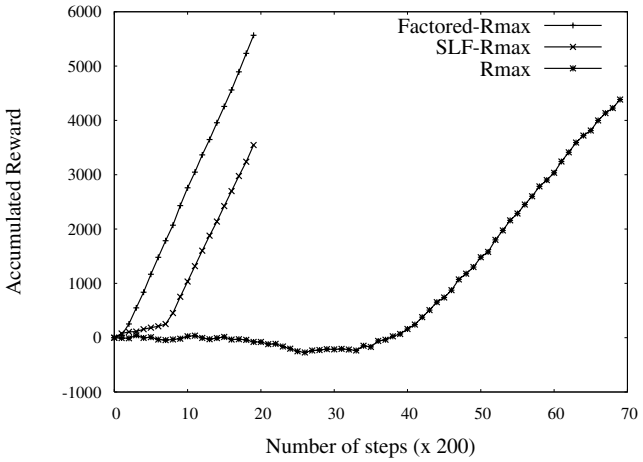

Figure 2: Cumulative reward on each timestep for the three algorithms: Factored Rmax (structure given), SLF-Rmax (structure learned) and Rmax (no structure).

References

Abbeel, P.; Koller, D.; and Ng, A. Y. 2006. Learning factor graphs in polynomial time and sample complexity. *JMLR*.

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

Brafman, R. I., and Tennenholtz, M. 2002. R-MAX—a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research* 3:213–231.

Degris, T.; Sigaud, O.; and Wuillemin, P.-H. 2006. Learning the structure of factored Markov decision processes in reinforcement learning problems. In *ICML-06: Proceedings of the 23rd international conference on Machine learning*, 257–264.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research* 13:227–303.

Guestrin, C.; Patrascu, R.; and Schuurmans, D. 2002. Algorithm-directed exploration for model-based reinforcement learning in factored MDPs. In *Proceedings of the International Conference on Machine Learning*, 235–242.

Kakade, S. M. 2003. *On the Sample Complexity of Reinforcement Learning*. Ph.D. Dissertation, Gatsby Computational Neuroscience Unit, University College London.

Kearns, M. J., and Koller, D. 1999. Efficient reinforcement learning in factored MDPs. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, 740–747.

Kearns, M. J., and Singh, S. P. 2002. Near-optimal reinforcement learning in polynomial time. *Machine Learning* 49(2–3):209–232.

Strehl, A. L.; Li, L.; and Littman, M. L. 2006. Incremental model-based learners with formal learning-time guarantees. In *UAI-06: Proceedings of the 22nd conference on Uncertainty in Artificial Intelligence*, 485–493.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. The MIT Press.