Expediting RL by Using Graphical Structures (Short Paper)

Peng Dai
Dept of Computer Science and Engineering
University of Washington
Seattle, WA 98195
daipeng@cs.washington.edu

Alexander L. Strehl
Yahoo! Research
New York, 10018
strehl@yahoo-inc.com

Judy Goldsmith
Dept of Computer Science
University of Kentucky
Lexington, KY 40506-0046
goldsmit@cs.uky.edu

ABSTRACT

The goal of Reinforcement learning (RL) is to maximize reward (minimize cost) in a Markov decision process (MDP) without knowing the underlying model *a priori*. RL algorithms tend to be much slower than planning algorithms, which require the model as input. Recent results demonstrate that MDP planning can be expedited, by exploiting the graphical structure of the MDP. We present extensions to two popular RL algorithms, Q-learning and RMax, that learn and exploit the graphical structure of problems to improve overall learning speed. Use of the graphical structure of the underlying MDP can greatly improve the speed of planning algorithms, if the underlying MDP has a nontrivial topological structure. Our experiments show that use of the *apparent* topological structure of an MDP speeds up reinforcement learning, even if the MDP is simply connected.

1. INTRODUCTION

Given a set of states, a set of actions, an initial state and a set of goal states, classical planning finds a sequence of actions that proceeds from the initial state to a goal state while minimizing cost. Decision theoretic planning [2] is a powerful extension that introduces outcome uncertainty.

Markov decision processes are a widely used model for AI researchers to represent decision theoretic planning problems. Given an MDP model, a planner finds a solution that has the optimal or at least acceptable cost. Classical MDPs solvers such as value iteration [1] use dynamic programming. This assumes the model is known. In reinforcement-learning (RL), the MDP environment is initially unknown, so dynamic programming is not immediately applicable.

There are two main approaches to RL, *model-free learning* and *model-based learning* (or simply *model-learning*) [12]. Model-free algorithms learn a value function or policy directly from the data, while model-based algorithms first construct an MDP model that they then use to reason about future actions and costs.

We show two basic RL algorithms can be made faster and more practical by learning and exploiting knowledge of the underlying graphical structure of environments. By examining the topological structure of the MDP's *reachability graph* rooted at the initial state, algorithms that use dynamic programming techniques can be modified to find near-optimal policies more quickly in many MDPs.

The contribution of our paper is three-fold. We show, in detail, how two representative reinforcement learning algorithms, Q-learning (model-free) and RMax (model-based), can be modified to use the underlying graphical MDP structure. We discuss how this method can be extended to current and future RL algorithms. Finally, we provide extensive empirical evaluation of our algorithms in comparison with the old algorithms. The experiments show that graphical-structure analysis significantly benefits RL algorithms.

Cite as: Expediting RL by Using Graphical Structures (Short Paper), Peng Dai, Alexander L. Strehl and Judy Goldsmith, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal,pp. 1325-1328.

2. BACKGROUND

A *scenario based MDP* is a six-tuple $\langle S, A, T, C, s_0, G\rangle$. S is a finite set of system states. A is a finite set of actions. $T_a(s'|s)$ is the probability of the system changing from state s to s' by performing action a. $C(s,a)$ is the instantaneous cost of performing action a at state s. $s_0 \in S$ is the initial state and $G \subseteq S$ is a set of goal states. We will use "MDP" for "scenario based MDP" for the rest of the paper.

Given an MDP, we define a *policy* $\pi : S \rightarrow A$ to be a mapping from the state space to the action space. A *value function* V^π for policy π, $V^\pi(s) : S \rightarrow \mathbf{R}$ denotes the value of the total expected cost starting from state s and following the policy π: $V^\pi(s) = C(s,\pi(s)) + \sum_{s' \in S} T_{\pi(s)}(s'|s)V^\pi(s')$. A policy π_1 dominates another policy π_2 if $V_{\pi_1}(s) \leq V_{\pi_2}(s)$ for all $s \in S$. An *optimal policy* π^* is a policy that is not dominated by any other policy, and is the optimal solution of the MDP. The value function of the optimal policy is called the *optimal value function* $V^*(\cdot)$. Bellman [1] showed that $V^*(\cdot)$ can be calculated by solving a system of linear equations in the form

$$V^*(s) = min_{a \in A(s)}[C(s,a) + \sum_{s' \in S} T_a(s'|s)V^*(s')]. \quad (1)$$

Equation 1 is also know as the *Bellman equation*. Using the Bellman equation as an assignment operator over a particular state is denoted as a *Bellman backup*. Bellman backups are the basic operations of dynamic programming, a technique of solving MDPs by calculating their optimal value functions.

Dynamic programming works directly in value function space. It backs up the value of states according to some order, until a time when further backups would result in only a very small change to the value function. We use the term *converge* loosely and informally to mean that the learned value function is sufficiently close to the optimal value functions. The simplest variant of value iteration [1], for example, initializes the value functions arbitrarily, and updates its value function by applying Bellman backups on every state in a fixed order. The algorithm halts when the largest change

in value during the most recent iteration is smaller than a threshold. Once the optimal value function is sufficiently approximated, a near-optimal policy, π, is easily extracted by choosing an action for each state:

$$\pi(s) = argmin_{a \in A(s)}[C(s,a) + \sum_{s' \in S} T_a(s'|s)V(s')].$$

Topological value iteration (TVI) [4] is a recent dynamic programming MDP algorithm . It makes use of the graphical structure of MDPs to perform Bellman backups in a better order. TVI first constructs a directed graph G from an MDP: the vertices of G are the states of the MDP, and the directed edges are state transitions. If the probability $T_a(s'|s) > 0$, then the edge $s \rightarrow s'$ is in G. TVI then computes the strongly connected components (SCCs) of G and their topological order. It solves every connected component sequentially by value iteration, according to this order. TVI outperforms VI significantly in MDP domains that have a reasonable number of SCCs.

3. MODEL-FREE LEARNING

Previously, we discussed how to obtain a near-optimal value function and policy for an MDP assuming we already have a *model*. The model consists of the cost function C and transition function T. In the reinforcement learning setting, we want to find a near-optimal value function and policy when the model is not initially provided. RL algorithms interact with the environment to get approximations of the model, and therefore solve the MDP.

Q-learning [13, 14] is a standard RL algorithm for MDPs. The algorithm maintains *Q-values* for each state action pair. $Q^*(s,a) = C(s,a) + \sum_{s' \in S} T_a(s'|s) min_{a'} Q^*(s',a')$. $Q^*(s,a)$ stands for the minimum expected cost of being in state s, applying action a, and then following the optimal policy. Thus, the optimal value function of s is the minimum Q-value with respect to s, $V^*(s) = min_a Q^*(s,a)$.

In MDPs, Q-learning initiates exploitation trials from the initial state. In each step of the trial, an action a is chosen for the current state s, which transitions the learning agent stochastically to s' according to the (unknown) transition function. A cost $c(s,a)$ is sensed, and $Q(a,s)$ is updated by $Q(s,a) = Q(s,a) + \alpha(c(s,a) + min_{a'} Q(s',a') - Q(s,a))$, where α is the *learning factor*, which it is often decreased as the time passes. Updating a Q-value is called a *Q-backup*.

Q-learning is very powerful, and is guaranteed to converge to an optimal policy, albeit sometimes slowly. One weakness is that it uses the same learning strategies for every MDP. The intuition behind our **topological Q-learning (TQL)** algorithm comes from TVI. TQL has two phases. The first phase is the initial learning phase. Here, we learn graphical information as well as Q-values. We initiate trials from the initial state the same way as Q-learning. As well as updating the value function for state-action pairs encountered along the trials, we record all predecessor-successor pairs visited during those trials. In other words, we mark all the visited edges of G. After a certain number (x) of trials, we use the recorded edge information to construct a directed graph, the *reachability graph,* G_R. Notice that the reachability graph is by no means guaranteed to be identical to the real G, since the trials might not visit all edges or even all states. However, if the learning process is sufficiently long, the information of learned state-action pairs is sufficient to solve the original MDP.

Given the reachability graph, we apply Kosaraju's algorithm to find the SCCs of G_R and their topological order. In the second phase, we choose one component at a time according to this order, pick one state from this component, and initiate trials from that state until the current component is converged. These trials are slightly different from trials of Q-learning. In Q-learning, a trial terminates only when a goal state is encountered. But trials of the second phase TQL finish when they run into a goal state or get into a state belonging to a component whose topological order is larger than the current one. This is because when a component is converged, all its states are converged, and we do not back up converged states. So if a later trial reaches converged states, we stop it. In each component, we initiate trials from the same state, since every other state in the component is reachable from this state. If we do enough trials, every state in that component gets backed up sufficiently.

Suppose we are asked to provide an online RL agent that takes advantage of the topological structure. We outline a simple extension to TQL to achieve this goal. When TQL learns that a transition from state s to s' is possible, it stores this fact in its reachability graph. With little additional overhead, we could store each experience-tuple (s,a,c,s') that is observed by agent, and link each of these tuples to the state s in our reachability graph.[1] Then, in the second phase, instead of initiating more trials from various states according to the topological ordering, we could simply run Q-learning over our saved experience-tuples from those states (and their outgoing neighbors in the reachability graph). This method can be viewed as a version of the "experience replay" algorithm [12] that takes advantage of learned topology.

One problem with the experience-replay approach described above is that storing every experience-tuple is memory intensive. An alternative approach is to maintain and update an approximate model of the underlying MDP.[2] After this, we can initiate Q-learning trials from any state by simulating them in our model. Alternatively, we could solve the model directly. This approach is developed in full detail in the next section.

4. MODEL-BASED LEARNING

Model-based RL algorithms use the agent's experience to estimate the system dynamics (transitions and costs) of the underlying MDP. It is straightforward to compute the maximum-likelihood model of the cost and transition distributions for each state-action pair. For instance, if we've seen n_1 transitions from state s to state s' after action a, out of n_2 total transitions from state s after action a, we would estimate the unknown transition probability $T_a(s'|s)$ by $\hat{T}_a(s'|s) = n_1/n_2$. As the agent gains experience over the state-action space, its model converges to the true MDP. Once the agent estimates the model, it can then solve the model using any MDP planner, and act according to an optimal policy. Unfortunately, when little experience has been gathered, the empirical model may be inaccurate, and resulting policies are suboptimal.

Several effective model-based algorithms have been developed, such as E^3 [7], RMax [3], and MBIE [11]. These algorithms estimate a model and its uncertainty. They use their models to obtain either the best known cost (exploitation) or knowledge that will reduce model uncertainty (exploration). The RMax algorithm is a model-based algorithm that has formal guarantees on its learning time [3, 6]. Therefore, we use it as a representative model-based RL algorithm. We describe RMax and discuss how to augment it to take advantage of the MDP's graphical structure. This very simple modification brings vast improvement.

The MDP model used by **RMax** contains the empirical transi-

[1]Here s' is the state reached and c is the immediate cost of taking action a at s.

[2]The Q-values computed by the experience replay algorithm converge to the optimal Q-values of the approximate model.

tion and reward distributions *only for those state-action pairs that have been experienced by the agent at least m times*, for some exploration parameter m. The transition distribution for other state-action pairs is a self loop, and the cost for those state-action pairs is 0, the minimum possible. The intuition is that the transition and cost estimates for those state-action pairs that have not been tried m times are likely to be inaccurate. Instead of using past experience to compute a model for these state-action pairs, we make them minimally costly in RMax's model. By choosing m carefully, RMax learns a near-optimal policy in polynomial time [3, 6].

Here we present an extension of RMax, **Topological RMax (TRMax)**. In RMax, whenever a new state-action pair (s, a) has been visited at least m times, we gather all the other *relevant* state-action pairs, the state-action pairs (s', a') that have the same property, and perform value iteration over them. Like TQL, TRMax has two phases. The first is the same as RMax, except we also remember the visited successor-predecessor pairs. After x trials, we compute the SCCs of the current reachability graph G_R as well as their topological order, then enter the second phase. From then on, when a state-action pair (s, a) has been visited m times, for a state-action pair (s', a') to be relevant, we require (s', a') to have been visited at least m times, and s' must belong to a component that has a higher topological order than the component of s in G_R. We use topological value iteration in solving the new model.

We originally extended RMax by recomputing the SCCs of the reachability graph and the topological ordering each time a new state-action pair was visited m times. We discarded that approach since constructing the SCCs of a directed graph is costly in practice. One possible improvement is to update the SCCs and topological ordering *incrementally* [10]. The overhead required may limit its practicality, but we plan to test this.

5. EXPERIMENTS

RL algorithms often do not have a well defined stopping criterion. During our experiments we kept a running average of the (estimated) value of the initial state. When the most recent value was sufficiently close to the long-term average, we terminated the experiment.

Any implementation of RMax must choose a technique for solving its model and this choice will affect the computational complexity of the algorithm. For our experiments, we used value iteration.

Our topological RL algorithms are based on the reachability graph that is known when we call Kosaraju's algorithm after x trials. What is a reasonable choice for x?

The *influence* of a state s with respect to a policy π, $I^\pi(s)$, is the expected number of times that state is visited in a trial following policy π [9]. Since any trial originates from the initial state s_0, the influences of s_0 is 1. Similarly, $\sum_{g \in G} I^\pi(g) = 1$. When $I^\pi(s) < 1$, it is the probability of s being visited in the exploitation trial. The influence of a state s with respect to the optimal policy is called the *optimal influence* $I^*(s)$.

$$I^\pi(s) = \sum_{s' \in S, a = \pi(s')} T_a(s|s') I^\pi(s'),$$

$$I^*(s) = \sum_{s' \in S, a = \pi^*(s')} T_a(s|s') I^*(s').$$

The influence measures the effect that changing the value of s will have on the value of s_0.

THEOREM 1. *If a state has an optimal influence of at least ϵ, then with probability $p = 1 - (1 - \epsilon)^t$, the optimal policy will visit it at least once in t trials. (In particular, when $\epsilon = 10^{-6}$, $t = 10,000$, $p = 0.99$.)*

PROOF. From the definition of $I^*(s)$, the probability that state s is *not* visited by a trial is $1 - I^*(s)$. Given t independent trials, the probability that s is not visited in any of them is $(1 - I^*(s))^t$, so the probability of s being visited at least once is $1 - (1 - I^*(s))^t$. By hypothesis, $I^*(s) \geq \epsilon$, so with probability $p = 1 - (1 - \epsilon)^t$, s should be visited at least once in t trials. □

We used $x = 10,000$ in our experiments. When we called Kosaraju's algorithm, states that were not visited in those x trials were ignored. Theoretically, we know from the above theorems that they have very small probabilities of making any real difference to the ultimate $V^*(s)$.

We tested Q-learning (QL), Topological Q-learning (TQL), RMax, and Topological RMax (TRMax). Each algorithm was implemented in C, and executed on the same Intel Pentium 4 1.50GHz processor with 512M main memory and a cache of 256kB.

$\|S\|$	5000		10000		1000		2000	
n_l	QL	TQL	QL	TQL	RMax	TRMax	RMax	TRMax
10	21.72	15.25	50.31	27.93	28.14	9.47	117.87	35.78
20	17.68	11.41	38.55	19.94	30.34	10.33	122.72	36.76
30	13.80	9.03	36.32	18.32	28.27	9.40	81.96	22.41
40	16.66	10.32	32.16	17.83	26.45	8.80	95.60	26.73
50	11.68	7.96	38.99	20.70	21.41	6.80	100.82	28.31
60	11.52	7.19	36.30	18.20	23.23	7.46	34.67	15.96
70	11.21	6.77	34.67	15.96	21.77	7.11	87.97	23.64
80	11.91	7.46	36.26	17.50	22.32	7.08	83.98	22.46
90	14.72	9.13	31.76	15.29	24.63	7.01	79.13	21.48
100	12.51	7.78	34.42	19.06	21.45	6.76	82.41	22.19

Table 1: Convergence time (seconds) of learning algorithms on MDPs m_a=5 and m_s = 10 with various layer numbers

domain	$\|S\|$	QL	TQL	RMax	TRMax
RMDP	1000	3.67	4.09	50.05	50.07
racetrack	1849	3.68	3.14	162.39	109.72
RMDP	2000	17.60	17.07	236.89	140.61
RMDP	4000	13.01	12.85	1043.83	576.53
racetrack	5566	24.90	23.82	976.84	279.65
RMDP	10000	43.55	37.83	-	3566.59
racetrack	21371	139.42	160.84	-	-
racetrack	50077	1443.17	1311.60	-	-

Table 2: Convergence time (seconds) of four algorithms on single connected component domains

We first used "layered" MDP domains,[3] [4], and larger problems for QL and TQL, which are usually faster than RMax and TRMax. Each layered MDP configuration is a four-tuple $\langle |S|, m_a, m_s, n_l \rangle$, where $|S|$ is the size of the MDP, m_a the maximum number of actions of each state, m_s the maximum number of successors of a state-action pair, n_l the number of layers. We fixed $|S|$, m_a=10, m_s=5, and varied n_l. For each configuration, we ran 20 MDPs, and averaged their statistics. For each problem, we measured the convergence time, the time taken to get an optimal policy, and the *deviation* of the calculated policy, the difference between the values $V^*(s_0)$ computed by RL algorithms and by value iteration. Convergence times are listed in Table 1. All the deviations in our experiments were $O(10^{-2})$, so were not listed. Looking at the table, we first notice that our topological learning algorithms converged faster than their basic algorithms. Comparing the left and right table, we also find that TRMax achieved a bigger speedup ratio over RMax compared to TQL over QL. This shows that model-based

[3]The "layered" MDPs are nonrepresentational MDPs with multiple SCCs.

learning benefited more from the graphical structure learning. Another interesting phenomenon is that as the number of layers increased, the running time of all the learning algorithms decreased. This is the opposite to the performance curve of dynamic programming approaches reported in [4].

Using TQL, we solved an MDP with 20,000 states and 100 layers within 1 minute, instead of more than 3 minutes by QL. We also solved an MDP with 4,000 states and 50 layers by TRMax within 2 minutes rather than over 6 minutes using RMax. The constant factor speedup shows that topological RL indeed widens the applicability of RL.

Topological value iteration reduces to value iteration when an MDP is strongly connected. We want to investigate if this is also the case for our topological learning algorithms. In this set of experiments, we used strongly-connected MDP problems. In Table 2, we listed the convergence times of algorithms on eight such problems. Random MDP was abbreviated as RMDP. The cut-off time was set at 90 minutes.

The convergence times of TQL were sometimes slower than QL. In those few cases, however, the termination time increased by at most 15%. Interestingly, TQL ran slightly faster than QL on three random MDP problems and two racetrack problems. This phenomenon is more distinct in the comparison between TRMax and RMax.

For the biggest racetrack problem we tested that RMax and TRMax can solve, TRMax was more than twice as fast as RMax, and consistently faster than RMax except for the smallest problem. This is counter-intuitive, since TRMax behaves like RMax except that it uses additional computation by calling Kosaraju's algorithm. The reason for these results follow. First, the *solution graph* of an MDP, containing the set of states and transitions that can be reached from s_0 using the optimal policy, has many fewer edges than G, so may contain multiple connected components. For our problems, the number of SCCs in the solution graph of two smaller racetrack problems are 546 and 1751 respectively. Similar observations were reported in the evaluation of policies for *partially observable MDPs* [5] (on page 117). In problems where a few actions are obviously better than others, the learning algorithm verifies their optimality quickly. The following trials continue to take these actions. Thus, some suboptimal action transitions might never be traversed. Our reachability graph, G_R, is built on the edges visited in the trials, so it skips unvisited suboptimal action transitions. Basically, backing up a state s is meaningful only when the backup is driven by a value change of the *descendants* of s (the set of states reachable from s) in the solution graph, because such a change might potentially change the value of $V^*(s_0)$. Since G_R skips a lot of edges that are not in the solution graph, most of the backups skipped by TRMax and not by RMax are necessary. So TRMax runs faster than RMax on strongly-connected MDPs.

6. RELATED WORK

The idea of performing value iteration on connected components in their topological order is not new. Our main contribution is to extend its applicability to the learning setting. The procedure described above is roughly outlined (on page 75) in the paper by [2]. It is streamlined and fully developed into the TVI algorithm and analyzed in full by [4].

In Prioritized Sweeping [8], states are prioritized according to their absolute Bellman error and backed up in priority order. Consider an MDP with connected components C_1, C_2, and C_3, connected in a chain, and states s_1, s_2, and s_3 in those components, respectively. Suppose that for actions a, a', and a'', $T_a(s_3|s_1) > 0$, $T_{a'}(s_3|s_2) > 0$, and $T_{a''}(s_2|s_1) > 0$. Suppose that the priority of backing up s_1 is always higher than the priority of s_2. When s_3 is backed up, s_1's value will be recomputed, and then s_2's value, which change s_1's. The situation is more complex when more components proceeds C_1. In TVI, the value for state s_3 is computed exactly (or closely approximated) before it is used to compute the values of s_2 and s_1, so it saves a lot of premature backups on s_2 and s_1. Wingate and Seppi [15] extended the notion of Prioritized Sweeping to General Prioritized Solvers. They consider a variety of prioritization schemes, and introduce the notion of partition. They do not, however, mention partitions via SCCs. They discuss topological order on vertices in a cyclic graph, and focus on approximating a topological order. Within a connected component, it might be possible to use one of their priority metrics to improve TVI.

7. CONCLUSION

We propose a practical method to speed up RL approaches for MDPs. By learning successor-predecessor information of MDP models during learning trials, we are able to construct a reachability graph that restores the dominating graphical structures of the original MDP. Using the topological order of SCCs in this reachability graph can help us either initiate useful future trials (model-free learning), or perform backups wisely (model-based learning). On all the problems tested, TQL and TRMax consistently outperformed their nontopological counterparts by a constant factor. We proved that it is safe to only consider the reachability graph instead of the original MDP as long as our initial learning is sufficient. Therefore, the scope of the problems solvable by these algorithms has been enlarged.

8. REFERENCES

[1] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

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

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

[4] P. Dai and J. Goldsmith. Topological value iteration algorithm for Markov decision processes. In *Proc. IJCAI-07*, pages 1860–1865, 2007.

[5] E. Hansen. *Finite Memory Control of Partially Observable Systems*. PhD thesis, University of Massachusetts, Amherst, 1998.

[6] S. M. Kakade. *On the sample complexity of reinforcement learning*. PhD thesis, Gatsby Computational Neuroscience Unit, University College London, 2003.

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

[8] A. Moore and C. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13:103–130, 1993.

[9] R. Munos and A. Moore. Influence and variance of a Markov chain : Application to adaptive discretization in optimal control. In *Proc. of IEEE Conference on Decision and Control*, 1999.

[10] D. J. Pearce and P. H. Kelly. A dynamic topological sort algorithm for directed acyclic graphs. *ACM J. of Experimental Algorithmics*, 11:1.7, 2007.

[11] A. L. Strehl and M. L. Littman. A theoretical analysis of model-based interval estimation. In *Proc. of ICML-05*, pages 856–863, 2005.

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

[13] C. J. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, UK, 1989.

[14] C. J. Watkins and P. Dayan. Q-Learning. *Machine Learning*, 8(3-):279–292, 1992.

[15] D. Wingate and K. D. Seppi. Prioritization methods for accelerating MDP solvers. *J. of Machine Learning Research*, 6:851–881, 2005.