Topological Value Iteration Algorithm for Markov Decision Processes

Peng Dai and **Judy Goldsmith**
Computer Science Dept.
University of Kentucky
773 Anderson Tower
Lexington, KY 40506-0046

Abstract

Value Iteration is an inefficient algorithm for Markov decision processes (MDPs) because it puts the majority of its effort into backing up the entire state space, which turns out to be unnecessary in many cases. In order to overcome this problem, many approaches have been proposed. Among them, LAO*, LRTDP and HDP are state-of-the-art ones. All of these use reachability analysis and heuristics to avoid some unnecessary backups. However, none of these approaches fully exploit the graphical features of the MDPs or use these features to yield the best backup sequence of the state space. We introduce an algorithm named Topological Value Iteration (TVI) that can circumvent the problem of unnecessary backups by detecting the structure of MDPs and backing up states based on topological sequences. We prove that the backup sequence TVI applies is optimal. Our experimental results show that TVI outperforms VI, LAO*, LRTDP and HDP on our benchmark MDPs.

1 Introduction

State-space search is a very common problem in AI planning and is similar to graph search. Given a set of states, a set of actions, a start state and a set of goal states, the problem is to find a policy (a mapping from states to actions) that starts from the start state and finally arrives at some goal state. *Decision theoretic planning* [Boutilier, Dean, & Hanks, 1999] is an attractive extension of the classical AI planning paradigm, because it allows one to model problems in which actions have uncertain and cyclic effects. Uncertainty is embodied in that one event can leads to different outcomes, and the occurrence of these outcomes are unpredictable, although they are guided by some form of predefined statistics. The systems are cyclic because an event might leave the state of the system unchanged or return to a visited state.

Markov decision process (MDP) is a model for representing decision theoretic planning problems. *Value iteration* and *policy iteration* [Howard, 1960] are two fundamental dynamic programming algorithms for solving MDPs. However, these two algorithms are sometimes inefficient. They spend too much time backing up states, often redundantly. Recently several types of algorithms have been proposed to efficiently solve MDPs. The first type uses reachability information and heuristic functions to omit some unnecessary backups, such as RTDP [Barto, Bradke, & Singh, 1995], LAO* [Hansen & Zilberstein, 2001], LRTDP [Bonet & Geffner, 2003b] and HDP [Bonet & Geffner, 2003a]. The second uses some approximation methods to simplify the problems, such as [Guestrin *et al.*, 2003; Poupart *et al.*, 2002; Patrascu *et al.*, 2002]. The third aggregates groups of states of an MDP by features, represents them as factored MDPs and solves the factored MDPs. Often the factored MDPs are exponentially simpler, but the strategies to solve them are tricky. SPUDD [Hoey *et al.*, 1999], sLAO* [Feng & Hansen, 2002], sRTDP [Feng, Hansen, & Zilberstein, 2003] are examples. One can use prioritization to decrease the number of inefficient backups. Focused dynamic programming [Ferguson & Stentz, 2004] and prioritized policy iteration [McMahan & Gordon, 2005] are two recent examples.

We propose an improvement of the value iteration algorithm named Topological Value Iteration. It combines the first and last technique. This algorithm makes use of graphical features of MDPs. It does backups in the best order and only when necessary. In addition to its soundness and optimality, our algorithm is flexible, because it is independent of any assumptions on the start state and can find the optimal value functions for the entire state space. It can easily be tuned to perform reachability analysis to avoid backups of irrelevant states. Topological value iteration is itself not a heuristic algorithm, but it can efficiently make use of extant heuristic functions to initialize value functions.

2 Background

In this section, we go over the basics of Markov decision processes and some of the extant solvers.

2.1 MDPs and Dynamic Programming Solvers

An MDP is a four-tuple (S, A, T, R). S is the set of states that describe how a system is at a given time. We consider the system developing over a sequence of discrete time slots, or *stages*. In each time slot, only one event is allowed to take effect. At any stage t, each state s has an associated set of applicable actions A_s^t. The effect of applying any action is to make the system change from the current state to the next state at stage $t+1$. The transition function for each action, T_a:

$S \times S \rightarrow [0,1]$, specifies the probability of changing to state s' after applying a in state s. $R : S \rightarrow \mathbf{R}$ is the instant reward (in our formulation, we use C C, the instant cost, instead of R). A *value function* V, $V : S \rightarrow \mathbf{R}$, gives the maximum value of the total expected reward from being in a state s. The *horizon* of a MDP is the total number of stages the system evolves. In problems where the horizon is a finite number H, our aim is to minimize the value $V(s) = \sum_{i=0}^{H} C(s^i)$ in H steps. For infinite-horizon problems, the reward is accumulated over an infinitely long path. To define the values of an infinite-horizon problem, we introduce a discount factor $\gamma \in [0,1]$ for each accumulated reward. In this case, our goal is to minimize $V(s) = \sum_{i=0}^{\infty} \gamma^i C(s^i)$.

Given an MDP, we define a policy $\pi : S \rightarrow A$ to be a mapping from states to actions. An optimal policy tells how we choose actions at different states in order to maximize the expected reward. Bellman [Bellman, 1957] showed that the expected value of a policy π can be computed using the set of value functions V^π. For finite-horizon MDPs, $V_0^\pi(s)$ is defined to be $C(s)$, and we define V_{t+1}^π according to V_t^π:

$$V_{t+1}^\pi(s) = C(s) + \sum_{s' \in S} \{T_{\pi(s)}(s'|s) V_t^\pi(s')\}. \quad (1)$$

For infinite-horizon MDPs, the (optimal) value function is defined as:

$$V(s) = min_{a \in A(s)}[C(s) + \gamma \sum_{s' \in S} T_a(s'|s) V(s')], \gamma \in [0,1]. \quad (2)$$

The above two equations are named *Bellman equations*. Based on Bellman equations, we can use dynamic programming techniques to compute the exact value of value functions. An optimal policy is easily extracted by choosing an action for each state that contributes its value function.

Value iteration is a dynamic programming algorithm that solves MDPs. Its basic idea is to iteratively update the value functions of every state until they converge. In each iteration, the value function is updated according to Equation 2. We call one such update a *Bellman backup*. The *Bellman residual* of a state s is defined to be the difference between the value functions of s in two consecutive iterations. The *Bellman error* is defined to be the maximum Bellman residual of the state space. When this Bellman error is less than some threshold value, we conclude that the value functions have converged sufficiently. Policy iteration [Howard, 1960] is another approach to solve infinite-horizon MDPs, consisting of two interleaved steps: policy evaluation and policy improvement. The algorithm stops when in some policy improvement phase, no changes are made. Both algorithms suffer from efficiency problems. Although each iteration of each algorithm is bound polynomially in the number of states, the number of iterations is not [Puterman, 1994].

The main drawback of the two algorithm is that, in each iteration, the value functions of every state are updated, which is highly unnecessary. Firstly, some states are backed up before their successor states, and often this type of backup is fruitless. We will show an example in Section 3. Secondly, different states converge with different rates. When only a few states are not converged, we may only need to back up a subset of the state space in the next iteration.

2.2 Other solvers

Barto et al. [Barto, Bradke, & Singh, 1995] proposed an online MDP solver named *real time dynamic programming*. This algorithm assumes that initially the algorithm knows nothing about the system except the information on the start state and the goal states. It simulates the evolution of the system by a number of trials. Each trial starts from the start state and ends at a goal state. In each step of the trial, one greedy action is selected based on the current knowledge and the state is changed stochastically. During the trial, all the visited states are backed up once. The algorithm succeeds when a certain number of trials are finished.

LAO* [Hansen & Zilberstein, 2001] is another solver that uses heuristic functions. Its basic idea is to expand an explicit graph G' iteratively based on some type of best-first strategy. Heuristic functions are used to guide which state is expanded next. Every time a new state is expanded, all its ancestor states are backed up iteratively, using value iteration. LAO* is a heuristic algorithm which uses the *mean first passage* heuristic. LAO* converges faster than RTDP since it expands states instead of actions.

The advantage of RTDP is that it can find a good suboptimal policy pretty fast, but the convergence for RTDP is slow. Bonet and Geffner extended RTDP to labeled RTDP (LRTDP) [Bonet & Geffner, 2003b], and the convergence of LRTDP is much faster. In their approach, they mark a state s as *solved* if the Bellman residuals of s and all the states that are reachable through the optimal policy from s are small enough. Once a state is solved, we regard its value function as converged, so it is treated as a "tip state" in the graph. LRTDP converges when the start state is solved.

HDP is another state-of-the-art algorithm, by Bonet and Geffner [Bonet & Geffner, 2003a]. HDP not only uses a similar labeling technique to LRTDP, but also discovers the connected components in the solution graph of a MDP. HDP labels a component as solved when all the states in that component have been labeled. HDP expands and updates states in a depth-first fashion rooted at the start states. All the states belonging to the solved components are regarded as tip states. Their experiments show that HDP dominated LAO* and LRTDP on most of the racetrack MDP benchmarks when the heuristic function h_{min} [Bonet & Geffner, 2003b] is used.

The above algorithms all make use of start state information by constraining the number of backups. The states that are unreachable from the start state are never backed up. They also make use of heuristic functions to guide the search to the promising branches.

3 A limitation of current solvers

None the algorithms listed above make use of inherent features of MDPs. They do not study the sequence of state backups according to an MDP's graphical structure, which is the intrinsic property of an MDP and potentially decides the complexity of solving it [Littman, Dean, & Kaelbling, 1995]. For example, Figure 1 shows a simplified version of an MDP. For simplicity, we omit explicit action nodes, transition probabilities, and reward functions. The goal state is marked in the figure. A directed edge between two states means the second

state is a potential successor state when applying some action in the first state.

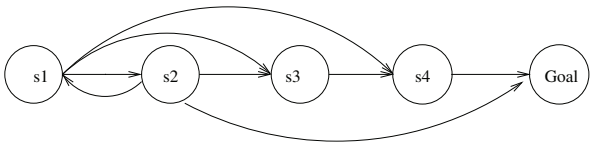

Figure 1: A simplified MDP

Observing the MDP in Figure 1, we know the best sequence to back up states is s_4, s_3, s_2, s_1, and if we apply this sequence, all the states except s_1 and s_2 only require one backup. However, not enough efforts of the algorithms mentioned above have been put to detect this optimal backup sequence. At the moment when they start on this MDP, all of them look at solving it as a common graph search problem with 5 vertices and apply essentially the same strategies as solving an MDP whose graphical structure is equivalent to a 5-clique, although this MDP is much simpler to solve than a 5-clique MDP. So the basic strategies of those solvers do not have an "intelligent" subroutine to distinguish various MDPs and to use different strategies to solve them. With this intuition, we want to design an algorithm that is able to discover the intrinsic complexity of various MDPs by studying their graphical structure and to use different backup strategies for MDPs with different graphical properties.

4 Topological Value Iteration

Our first observation is that states and their value functions are causally related. If in an MDP M, one state s' is a successor state of s after applying action a, then $V(s)$ is dependent on $V(s')$. For this reason, we want to back up s' ahead of s. The causal relation is transitive. However, MDPs are cyclic and causal relations are very common among states. How do we find an optimal backup sequence for states? Our idea is the following: We group states that are mutually causally related together and make them a *metastate*, and let these metastates form a new MDP M'. Then M' is no longer cyclic. In this case, we can back up states in M' in their reverse topological order. In other words, we can back up these big states in only one *virtual* iteration. How do we back up the big states that are originally sets of states? We can apply any strategy, such as value iteration, policy iteration, linear programming, and so on. How do we find those mutually causally related states?

To answer the above question, let us look at the graphical structure of an MDP first. An MDP M can be regarded as a directed graph $G(V, E)$. The set V has state nodes, where each node represents a state in the system, and action nodes, where each action in the MDP is mapped to a vertex in G. The edges, E, in G represent transitions, so they indicate the causal relations in M. If there is an edge e from state node s to node a, this means a is a candidate action for state s. Conversely, an edge e pointing from a to s' means, applying action a, the system has a positive probability of changing to state s'. If we can find a path $s \rightarrow a \rightarrow s'$ in G, we know that state s is causally dependent on s'. So if we simplify G by removing all the action nodes, and changing paths like $s \rightarrow a \rightarrow s'$ into directed edges from s to s', we get a causal relation graph G_{cr} of the original MDP M. A path from state s_1 to s_2 in G_{cr} means s_1 is causally dependent on s_2. So the problem of finding mutually causally related groups of states is reduced to the problem of finding the strongly connected components in G_{cr}.

We use Kosaraju's [Cormen *et al.*, 2001] algorithm of detecting the topological order of strongly connected components in a directed graph. Note that Bonet and Geffner [Bonet & Geffner, 2003a] used Tarjan's algorithm in detection of strongly connected components in a directed graph in their solver, but they do not use the topological order of these components to systematically back up each component of an MDP. Kosaraju's algorithm is simple to implement and its time complexity is only linear in the number of states, so when the state space is large, the overhead in ordering the state backup sequence is acceptable. Our experimental results also demonstrate that the overhead is well compensated by the computational gain.

The pseudocode of TVI is shown in Figure 2. We first use Kosaraju's algorithm to find the set of strongly connected components C in graph G_{cr}, and their topological order. Note that each $c \in C$ maps to a set of states in M. We then use value iteration to solve each c. Since there are no cycles in those components, we only need to solve them once. Notice that, when the entire state space is causally related, TVI is equivalent to VI.

Theorem 1 *Topological Value Iteration is guaranteed to converge to the optimal value function.*

Proof We first prove TVI is guaranteed to terminate in finite time. Since each MDP contains a finite number of states, it contains a finite number of connected components. In solving each of these components, TVI uses value iteration. Because value iteration is guaranteed to converge in finite time, TVI, which is actually a finite number of value iterations, terminates in finite time. We then prove TVI is guaranteed to converge to the optimal value function. According to the update sequence of TVI, at any point of the algorithm, the value functions of the states (of one component) that are being backed up only depend on the value functions of the components that have been backed up, but not on those of the components that have not been backed up. For this reason, TVI lets the value functions of the state space converge sequentially. When a component is converged, the value functions of the states can be safely used as tip states, since they can never be influenced by components backed up later.

A straightforward corollary to the above theorem is:

Corollary 2 *Topological Value Iteration only updates the value functions of a component when it is necessary. And the update sequence of the update is optimal.*

4.1 Optimization

In our implementation, we added two optimizations to our algorithm. One is reachability analysis. TVI does not assume any initial state information. However, given that information, TVI is able to detect the unreachable components and

Topological Value Iteration

TVI(MDP M, δ)
1 . $scc(M)$
2 . for $i \leftarrow 1$ to $cpntnum$;
3 . $S \leftarrow$ the set of states s where $s.id = cpntnum$
4 . $vi(S, \delta)$
vi(S: a set of states, δ)
5 . while $(true)$
6 . for each state $s \in S$
7 . $V(s) = min_{a \in A(s)}[C(s) + \gamma \sum_{s' \in S} T_a(s'|s)V(s')]$
8 . if (Bellman error is less than δ)
9 . return
scc(MDP M) (Kosaraju's algorithm)
10. construct G_{cr} from M by removing action nodes
11. construct the reverse graph G'_{cr} of G_{cr}
12. $size \leftarrow$ number of states in G_{cr}
13. for $s \leftarrow 1$ to $size$
14. $s.id \leftarrow -1$
15. // $postR$ and $postI$ are two arrays of length $size$
16. $cnt \leftarrow 1, cpntnum \leftarrow 1$
17. for $s \leftarrow 1$ to $size$
18. if $(s.id = -1)$
19. $dfs(G', s)$
20. for $s \leftarrow 1$ to $size$
21. $postR[s] \leftarrow postI[s]$
22. $cnt \leftarrow 1, cpntnum \leftarrow 1$
23. for $s \leftarrow 1$ to $size$
24. $s.id \leftarrow -1$
25. for $s \leftarrow 1$ to $size$
26. if $(s.id = -1)$
27. $dfs(G_{cr}, postR[s])$
28. $cpntnum \leftarrow cpntnum + 1$
29. return $(cpntnum, G_{cr})$
dfs(Graph G, s)
30. $s.id \leftarrow cpntnum$
31. for each successor s' of s
32. if $(s'.id = -1)$
33. $dfs(G, s')$
34. $postI[cnt] \leftarrow s$
35. $cnt \leftarrow cnt + 1$

Figure 2: Pseudocode of Topological Value Iteration

ignore them in the dynamic programming step. Reachability is computed by a depth first search. The overhead of this analysis is linear, and it helps us avoid considering the unreachable components, so the gains can well compensate for the trouble introduced. It is extremely useful when only a small portion of the state space is reachable. Since the reachability analysis is straightforward, we do not provide any pseudocode for it. The other optimization is the use of heuristic functions. Heuristic values can serve as a good starting point for value functions in TVI. In our program, we use the h_{min} heuristic from [Bonet & Geffner, 2003b]. Reachability analysis and the use of heuristics help strengthen the competitiveness of TVI. h_{min} replaces the expected future reward part of the Bellman equation by the minimum of such value. It is an admissible heuristic.

$$h_{min}(s) = min_a[C(s) + \gamma \cdot min_{s':T_a(s'|s)>0}V(s')]. \quad (3)$$

5 Experiment

We tested the topological value iteration and compared its running time against value iteration (VI), LAO*, LRTDP and HDP. All the algorithms are coded in C and properly optimized, and run on the same Intel Pentium 4 1.50GHz processor with 1G main memory and a cache size of 256kB. The operating system is Linux version 2.6.15 and the compiler is gcc version 3.3.4.

5.1 Domains

We use two MDP domains for our experiments. The first domain is a model simulating PhD qualifying exams. We consider the following scenario from a fictional department: To be qualified for a PhD in Computer Science, one has to pass exams in each CS area. Every two months, the department offers exams in each area. Each student takes each exam as often as he wants until he passes it. Each time, he can take at most two exams. We consider two types of grading criteria. For the first criterion, we only have pass and fail (and of course, untaken) for each exam. Students who have not taken and who have failed certain exam before have the same chance of passing that exam. The second criterion is a little trickier. We assign pass, conditional pass, and fail to each exam, and the probabilities of passing certain exams vary, depending on the student's past grade on that exam. A state in this domain is a value assignment of the grades of all the exams. For example, if there are five exams, *fail,pass,pass,condpass,untaken* is one state. We refer to the first criterion MDPs as $QE_s(e)$ and second as $QE_t(e)$, where e refers to the number of exams.

For the second domain, we use artificially-generated "layered" MDPs. For each MDP, we define the number of states, and partition them evenly into a number n_l of *layers*. We number these layers by numerical values. We allow states in higher numbered layers to be the successor states of states in lower numbered layers, but not vice versa, so each state has only a limited set of allowable successor states $succ(s)$. The other parameters of these MDPs are: the maximum number of actions each state can have is m_a, the maximum number of successor states of each action, m_s. Given a state s, we let the pseudorandom number generator of C pick the number of actions from $[1, m_a]$, and for each action, we let that action have a number of successor states in $[1, m_s]$. The states are chosen uniformly from $succ(s)$ together with normalized transition probabilities. The advantage of generating MDPs this way is that these layered MDPs contain at least n_l connected components.

There are actual applications that lead to multi-layered MDPs. A simple example is the game Bejeweled: each level is at least one layer. Or consider a chess variant without pawns, played against a stochastic opponent. Each set of pieces that could appear on the board together leads to (at least one strongly-connected component. There are other, more serious examples, but we know of no multi-layered standard MDP benchmarks. Examples such as race-track MDPs tend to have a single scc, rendering TVI no better than VI. (Since checking the topological structure of an MDP takes negligible time compared to running any of the solvers, it is

domain	$QE_s(7)$	$QE_s(8)$	$QE_s(9)$	$QE_s(10)$	$QE_t(5)$	$QE_t(6)$	$QE_t(7)$	$QE_t(8)$
$\|S\|$	2187	6561	19683	59049	1024	4096	16384	65536
$\|a\|$	28	36	45	55	15	21	28	36
$\#ofscc's$	2187	6561	19683	59049	243	729	2187	6561
$v^*(s_0)$	11.129919	12.640260	14.098950	15.596161	7.626064	9.094187	10.565908	12.036075
h_{min}	4.0	4.0	5.0	5.0	3.0	4.0	4.0	5.0
VI($h=0$)	1.08	4.28	15.82	61.42	0.31	1.89	10.44	59.76
LAO*($h=0$)	0.73	4.83	26.72	189.15	0.27	2.18	16.57	181.44
LRTDP($h=0$)	0.44	1.91	7.73	32.65	0.28	2.05	16.68	126.75
HDP($h=0$)	5.44	75.13	1095.11	1648.11	0.75	29.37	1654.00	2130.87
TVI($h=0$)	0.42	1.36	4.50	15.89	0.20	1.04	5.49	35.10
VI(h_{min})	1.05	4.38	15.76	61.06	0.31	1.87	10.41	59.73
LAO*(h_{min})	0.53	3.75	19.16	126.84	0.25	1.94	14.96	123.26
LRTDP(h_{min})	0.28	1.22	4.90	20.15	0.28	1.95	16.22	124.69
HDP(h_{min})	4.42	59.71	768.59	1583.77	0.95	30.14	1842.62	2915.05
TVI(h_{min})	0.16	0.56	1.86	6.49	0.19	0.98	5.29	30.79

Table 1: Problem statistics and convergence time in CPU seconds for different algorithms with different heuristics for the qual. exams examples ($\epsilon = 10^{-6}$)

easy to decide whether to use TVI.) Thus, we use our artificially generated MDPs for now.

5.2 Results

We consider several variants of our first domain, and the results are shown in Table 1. The statistics have shown that:

- TVI outperforms the rest of the algorithms in all the instances. Generally, this fast convergence is due to both the appropriate update sequence of the state space and avoidance of unnecessary updates.
- The h_{min} helps TVI more than it helps VI, LAO* and LRTDP, especially in the QE_s domains.
- TVI outperforms HDP, because our way of dealing with components is different. HDP updates states of all the unsolved components together in a depth-first fashion until they all converge. We pick the optimal sequence of backing up components, and only back up one of them at a time. Our algorithm does not spend time checking whether all the components are solved, and we only update a component when it is necessary.

We notice that HDP shows pretty slow convergence in the QE domain. That is not due to our implementation. HDP is not suitable for solving problems with large numbers of actions. Readers interested in the performance of HDP on MDPs with smaller action sets can refer to [Bonet & Geffner, 2003a].

The statistics of the performance on artificially generated layered MDPs are shown in Table 2 and 3. We do not include the HDP statistics here, since HDP is too slow in these cases. We also ignore the results on applying the h_{min} heuristic, since they display the same scale as not using the heuristic. For each element of the table, we take the average of running 20 instances of MDPs with the same configuration. Note that varying $|S|$, n_l, m_a, and m_s yields many MDP configurations. We present a few whose results are representative.

For the first group of data, we fix the state space to have size 20,000 and change the number of layers. Statistics in Table 2 show our TVI dominates others. We note that, as the layer number increases, the MDPs become more complex, since the states in large numbered layers have relatively small $succ(s)$ against m_s, therefore cycles in those layers are more common, so it takes greater effort to solve large numbered layers than small numbered ones. Not surprisingly, from Table 2 we see that when the number of layers increases, the running time of each algorithm also increases. However, the increase rate of TVI is the smallest (the rate of greatest against smallest running time of TVI is 2 versus 4 of VI, 3.5 of LAO*, and 2.3 of LRTDP). This is due to the fact that TVI applies the best update sequence. As the layer number becomes large, although the update of the large numbered layers requires more effort, the time TVI spends on the small numbered ones remains stable. But other algorithms do not have this property.

For the second experiment, we fix the number of layers and vary the state space size. Again, TVI is better than other algorithms, as seen in Table 3. When the state space is 80,000, TVI can solve the problems in around 12 seconds. This shows that TVI can solve large problems in a reasonable amount of time. Note that the statistics we include here represent the common cases, but were not chosen in favor of TVI. Our best result shows, TVI runs in around 9 seconds for MDPs with $|S|$=20,000, n_l=200, m_a=20, m_s=40, while VI needs more than 100 seconds, LAO* takes 61 seconds and LRTDP requires 64 seconds.

6 Conclusion

We have introduced and analyzed an MDP solver, topological value iteration, that studies the dependence relation of the value functions of the state space and use the dependence relation to decide the sequence to back up states. The algorithm is based on the idea that different MDPs have different graphical structures, and the graphical structure of an MDP intrinsically determines the complexity of solving that MDP. We notice that no current solvers detect this information and use it to guide state backups. Thus, they solve MDPs of the same problem sizes but with different graphical structure with

n_l	20	40	60	80	100	200	300	400	500	600
VI($h=0$)	19.482	17.787	12.310	24.538	31.684	40.782	46.554	52.248	64.706	77.658
LAO*($h=0$)	6.088	6.584	5.702	7.297	9.250	12.801	12.463	12.259	16.255	21.991
LRTDP($h=0$)	9.588	9.651	8.483	8.352	11.165	11.108	13.125	13.443	14.160	22.057
TVI($h=0$)	2.563	2.686	2.587	2.624	2.805	3.186	3.061	4.065	4.264	5.131

Table 2: Problem statistics and convergence time in CPU seconds for different algorithms on solving layered MDPs with different number of layers ($|s|$=20000, m_a=10, m_s=20, $\epsilon = 10^{-6}$)

$\|S\|$	10000	20000	30000	40000	50000	60000	70000	80000
VI($h=0$)	9.322	27.683	38.529	46.178	64.915	72.087	95.479	117.359
LAO*($h=0$)	3.056	6.201	10.561	13.663	21.409	23.461	29.248	38.746
LRTDP($h=0$)	3.903	8.814	10.725	20.708	27.156	53.773	49.954	50.730
TVI($h=0$)	1.212	2.589	4.055	5.571	7.133	8.626	10.307	11.969

Table 3: Problem statistics and convergence time in CPU seconds for different algorithms on solving layered MDPs with different state spaces (n_l=20, m_a=10, m_s=20, $\epsilon = 10^{-6}$)

almost the same strategies. In this sense, they are not "intelligent". Topological value iteration is proposed to solve this problem. It is guaranteed to find the optimal solution of a Markov decision process sequentially. Topological value iteration is a flexible algorithm, which can use the initial state information and apply reachability analysis. However, even without this information, TVI runs soundly and completely.

We coded up an academic example and our experimental results show that TVI outperforms not only VI, but also LAO*, LRTDP and HDP, state of the art algorithms. The reason is because our algorithm is able to detect the optimal sequence of updating each component and components are only updated when necessary. Without standard, layered benchmarks, we test our algorithm on artificially generated layered MDPs. Our results on these MDPs have shown that TVI is extremely useful in MDPs with many connected components. The complexity increase of TVI is not as great as other algorithms as the number of layers increase, which shows that TVI is very suitable for solving MDPs with layered structures.

References

[Barto, Bradke, & Singh, 1995] Barto, A.; Bradke, S.; and Singh, S. 1995. Learning to act using real-time dynamic programming. *JAI* 72:81–138.

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

[Bonet & Geffner, 2003a] Bonet, B., and Geffner, H. 2003a. Faster heuristic search algorithms for planning with uncertainty and full feedback. In Gottlob, G., ed., *Proc. of IJCAI-03*, 1233–1238. Acapulco, Mexico: Morgan Kaufmann.

[Bonet & Geffner, 2003b] Bonet, B., and Geffner, H. 2003b. Labeled RTDP: Improving the convergence of real-time dynamic programming. In Giunchiglia, E.; Muscettola, N.; and Nau, D., eds., *Proc. 13th ICAPS*, 12–21. Trento, Italy: AAAI Press.

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

[Cormen *et al.*, 2001] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, 2001.

[Feng & Hansen, 2002] Feng, Z., and Hansen, E. A. 2002. Symbolic heuristic search for factored Markov decision processes. In *Proc. of AAAI-05*.

[Feng, Hansen, & Zilberstein, 2003] Feng, Z.; Hansen, E. A.; and Zilberstein, S. 2003. Symbolic generalization for on-line planning. In *Proc. of UAI-03*, 209–216.

[Ferguson & Stentz, 2004] Ferguson, D., and Stentz, A. 2004. Focussed dynamic programming: Extensive comparative results. Technical Report CMU-RI-TR-04-13, Carnegie Mellon University, Pittsburgh, PA.

[Guestrin *et al.*, 2003] Guestrin, C.; Koller, D.; Parr, R.; and Venkataraman, S. 2003. Efficient solution algorithms for factored MDPs. *JAIR* 19:399–468.

[Hansen & Zilberstein, 2001] Hansen, E., and Zilberstein, S. 2001. LAO*: A heuristic search algorithm that finds solutions with loops. *JAI* 129:35–62.

[Hoey *et al.*, 1999] Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 1999. SPUDD: Stochastic planning using decision diagrams. In *Proc. of UAI-99*, 279–288.

[Howard, 1960] Howard, R. 1960. *Dynamic Programming and Markov Processes*. Cambridge, Massachusetts: MIT Press.

[Littman, Dean, & Kaelbling, 1995] Littman, M. L.; Dean, T.; and Kaelbling, L. P. 1995. On the complexity of solving Markov decision problems. In *Proc. of UAI-95*, 394–402.

[McMahan & Gordon, 2005] McMahan, H. B., and Gordon, G. J. 2005. Fast exact planning in Markov decision processes. In *Proc. of ICAPS-05*.

[Patrascu *et al.*, 2002] Patrascu, R.; Poupart, P.; Schuurmans, D.; Boutilier, C.; and Guestrin, C. 2002. Greedy linear value-approximation for factored Markov decision processes. In *Proc. of AAAI-02*, 285–291.

[Poupart *et al.*, 2002] Poupart, P.; Boutilier, C.; Patrascu, R.; and Schuurmans, D. 2002. Piecewise linear value function approximation for factored MDPs. In *Proc. of AAAI-02*, 292–299.

[Puterman, 1994] Puterman, M. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley, New York.