Focused Topological Value Iteration

Peng Dai Mausam Daniel S. Weld
Dept of Computer Science and Engineering
University of Washington
Seattle, WA-98195
{daipeng,mausam,weld}@cs.washington.edu

Abstract

Topological value iteration (TVI) is an effective algorithm for solving Markov decision processes (MDPs) optimally, which 1) divides an MDP into strongly-connected components, and 2) solves these components sequentially. Yet, TVI's usefulness tends to degrade if an MDP has large components, because the cost of the division process isn't offset by gains during solution. This paper presents a new algorithm to solve MDPs optimally, *focused topological value iteration* (FTVI). FTVI addresses TVI's limitations by restricting its attention to connected components that are *relevant for solving the MDP*. Specifically, FTVI uses a small amount of heuristic search to eliminate provably sub-optimal actions; this pruning allows FTVI to find smaller connected components, thus running faster. We demonstrate that our new algorithm outperforms TVI by an order of magnitude, averaged across several domains. Surprisingly, FTVI also significantly outperforms popular 'heuristically-informed' MDP algorithms such as LAO*, LRTDP, and BRTDP in many domains, sometimes by as much as two orders of magnitude. Finally, we characterize the type of domains where FTVI excels — suggesting a way to an informed choice of solver.

Introduction

Markov Decision Processes (MDPs) (Bellman 1957) are a powerful and widely-adopted formulation for modeling autonomous decision making under uncertainty. For instance, NASA researchers use MDPs to model the Mars rover planning problems (Bresina et al. 2002). MDPs are also used to formulate the military operations planning (Aberdeen, Thiébaux, and Zhang 2004) and coordinated multi-agent planning (Musliner et al. 2007), *etc.*

Classical dynamic programming algorithms, such as value iteration (VI), solve a Markov decision process optimally by iteratively updating the value of every state by a fixed order, one state per iteration. This is sometimes very inefficient, since it overlooks the graphical structure of a problem, which can provide vast information about the state dependencies. Recently, Dai and Goldsmith (2007) developed a new algorithm named topological value iteration (TVI), which performs an additional topological analysis of the MDP state space. TVI first divides an MDP into strongly connected components and then solves each strongly connected component sequentially in the topological order. Experimental results demonstrated significant performance gains over VI and other heuristic search algorithms in a specific kind of domain – one that has many small strongly connected components (*e.g.*, a chess game against a stochastic opponent (Dai and Goldsmith 2007)).

However, such a graphical structure is not present in many benchmark domains leaving TVI's performance no better (or often worse, due to the overhead of topological analysis) than other MDP algorithms. For instance, many domains (*e.g.*, Blocksworld) have reversible actions. For these domains all of the states connected by reversible actions end up being in one (large) strongly connected component, thus, reducing the benefit of TVI. In our work we wish to remove this limitation by observing that, while the complete MDP may be strongly connected, the subset of states relevant for solving the particular problem at hand may be divisible into smaller components.

We present a novel algorithm to solve MDPs optimally called *focused topological value iteration* (FTVI), which addresses the weaknesses of TVI. FTVI first performs a phase of heuristic search and eliminates provably sub-optimal actions found during the search. Then it builds a more informative graphical structure based on the remaining actions. We find that a very short phase of heuristic search is often able to eliminate many actions leading to an MDP structure that is amenable to efficient topology-based solutions.

We evaluate FTVI across several benchmark domains and find that FTVI outperforms TVI by significant margins. Surprisingly, we also find that FTVI outperforms state-of-the-art heuristic search algorithms in many domains. This is unexpected, since common wisdom dictates that heuristic guided search is much faster than all-state dynamic programming. To better understand this big improvement, we study the convergence speed of heuristic search algorithms on a few problem features. We discover two important features of problems that are hard for heuristic search algorithms: smaller number of goal states and long search distance to the goal. These features are commonly found in many domains, *e.g.*, Mountain car (Wingate and Seppi 2005) and Wet-floor (Bonet and Geffner 2006). We show that, in such domains, FTVI outperforms heuristic search by an order of magnitude on average, and sometimes by even two orders of magnitude.

Background

Markov Decision Processes

AI researchers typically use MDPs to formulate probabilistic planning problems. An MDP is defined as a four-tuple $\langle \mathcal{S}, \mathcal{A}, T, C\rangle$, where $\mathcal{S}$ is a discrete set of states, $\mathcal{A}$ is a finite set of all applicable actions, T is the transition matrix describing the domain dynamics, and C denotes the cost of action transitions.

The agent executes its actions in discrete time steps called *stages*. At each stage, the system is at one distinct state $s \in \mathcal{S}$. The agent can pick any action a from a set of *applicable action* $Ap(s) \subseteq \mathcal{A}$, incurring a cost of $C(s,a)$. The action takes the system to a new state s' stochastically, with probability $T_a(s'|s)$.

The *horizon* of an MDP is the number of stages for which costs are accumulated. For ease of illustration, we concentrate on a special set of MDPs called *stochastic shortest path* (SSP) problems.[1] The horizon in such an MDP is indefinite and the costs are accumulated with no discounting. There are an initial state s_0, and a set of sink *goal states* $\mathcal{G} \subseteq \mathcal{S}$. Reaching any one of $g \in \mathcal{G}$, terminates the execution. The cost of the execution is the sum of all costs along the path from s_0 to a goal g.

To solve the MDP we need to find an *optimal policy* ($\pi^* : \mathcal{S} \rightarrow \mathcal{A}$), a probabilistic execution plan that reaches a goal state with the minimum expected cost. We evaluate any policy π by a *value function*.

$$V^\pi(s) = C(s,\pi(s)) + \sum_{s' \in \mathcal{S}} T_{\pi(s)}(s'|s)V^\pi(s').$$

Any optimal policy must satisfy the following system of *Bellman equations*:

$$\begin{aligned} V^*(s) &= 0 \quad \text{if } s \in \mathcal{G} \text{ else} \\ V^*(s) &= \min_{a \in Ap(s)} [C(s,a) + \sum_{s' \in \mathcal{S}} T(s'|s,a)V^*(s')]. \end{aligned} \tag{1}$$

The corresponding optimal policy can be extracted from the value function:

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

Q-value is a key notion in MDP algorithms. Given an implicit optimal policy π^*, in the form of its optimal value function $V^*(\cdot)$, the Q-value of a state-action pair (s,a) is defined as the value of state s, if an immediate action a is performed, followed by π^* afterwards. More concretely,

$$Q^*(s,a) = C(s,a) + \sum_{s' \in \mathcal{S}} T_a(s'|s)V^*(s'). \tag{2}$$

Therefore, the optimal value function can be rewritten as:

$$V^*(s) = min_{a \in Ap(s)} Q^*(s,a).$$

[1]Despite its simplicity, SSP is a general MDP representation. Any infinite horizon discounted reward MDP can be easily converted to an undiscounted SSP (Bertsekas and Tsitsiklis 1996).

Dynamic Programming

Most optimal MDP algorithms are based on dynamic programming. Its usefulness was first proved by a simple yet powerful algorithm named *value iteration* (Bellman 1957). Value iteration first initializes the value function arbitrarily. Then, the values are updated iteratively using an operator called *Bellman backup* to create successively better approximations per state per iteration. Value iteration stops updating when the value function converges (one future backup can change a state value by at most ϵ, a pre-defined threshold).

Value iteration converges to the optimal value function in time polynomial in $|\mathcal{S}|$ (Littman, Dean, and Kaelbling 1995), yet in practice it is usually inefficient, since it blindly performs backups over the state space iteratively, often introducing many unnecessary backups over the state space.

Topological Value Iteration Topological value iteration (TVI) (Dai and Goldsmith 2007) is an enhancement of value iteration, which solves an MDP problem by using the problem's graphical structure wisely. Given an MDP, TVI first builds a directed reachability graph G_R, where G has one vertex per state $s \in \mathcal{S}$. A directed edge from vertex s_1 to s_2 exists if there is an action a such that $T_a(s_2|s_1) > 0$. TVI then finds all the strongly connected components of G_R, and the topological order of the components. Later, it solves every connected component individually, by value iteration, according to their topological order. By decomposing an MDP into smaller components, TVI's convergence can be much faster than VI. Results (Dai and Goldsmith 2007) show that TVI outperforms VI and state-of-the-art heuristic search algorithms when an MDP has many small components.

Heuristic Search To improve the efficiency of dynamic programming, researchers (Barto, Bradtke, and Singh 1995; Hansen and Zilberstein 2001; Bonet and Geffner 2003b; 2006; McMahan, Likhachev, and Gordon 2005; Smith and Simmons 2006) have explored various ideas from traditional heuristic-guided search, and have consistently demonstrated its usefulness for MDPs. The basic idea of heuristic search is to expand an action only when necessary, and leads to a more conservative backup strategy. This strategy helps save a lot of unnecessary backups.

Heuristic search algorithms have two main features: (1) values of the state space are initialized by an admissible and consistent heuristic function. Note that dynamic programming algorithms such as VI and TVI can also take advantage of initial heuristic values as an informative bootstrap, but do not require the heuristics to be admissible or consistent to guarantee optimality. (2) The search is limited to states that are reachable from the initial state. Given the heuristic value, heuristic search generates a running *greedy policy* – the best policy by one-step lookahead given the current value function, as well as the *greedy policy graph* – a subset of the original MDP that contains all states that are reachable from the initial state through the current greedy policy and corresponding transitions. The algorithm performs a series of heuristic searches, until states on the greedy policy graph converge. A search typically starts from the initial state, and expands along a greedy action, either deterministically or stochastically. Visited states have their values backed-up during the search.

Different heuristic search algorithms use different search strategies, and therefore perform Bellman backups in different orders. For example, searches in real-time dynamic programming (RTDP) and its variants (Barto, Bradtke, and Singh 1995; Bonet and Geffner 2003b; McMahan, Likhachev, and Gordon 2005; Smith and Simmons 2006) occur in the form of execution trials. An execution trial is a path from s_0 to any goal state, where the next state is chosen stochastically based on the greedy action of the current state. States are backed-up immediately when they are visited. Searches in improved LAO* (ILAO*) (Hansen and Zilberstein 2001) traverse the entire greedy policy graph. It starts from s_0, and expands frontier states in the depth-first manner. Visited states are backed-up in the post-order.

Focused Topological Value Iteration

Heuristic search is a powerful solution technique. It successfully concentrates computation, in the form of backups, on states and transitions that are more likely to be part of an optimal policy. However, heuristic search iteratively evaluates the whole greedy policy, which itself can be time consuming.

On the other hand, topological value iteration is a pure dynamic programming algorithm and does not remove any state from consideration. It improves the performance of value iteration when an MDP has many small connected components. However, after careful analysis over many MDP domains, we observe that many MDPs do not have evenly distributed connected components. This is due to the following reason: a state can have many actions, most of which are sub-optimal. These sub-optimal actions, although not part of an optimal policy, may lead to connectivity between a lot of states. For example, domains like Blocksworld have reversible actions. Due to these actions most states get connected bidirectionally. As a result, states connected by reversible actions end up forming a large connected component, making TVI slow.

Continuing with the example we note that an optimal policy seldom contains an action and its reverse, since that will typically make the plan more costly. If we know that an action is guaranteed to be better than its reverse, we can eliminate the reverse and break connected components. In general, we can eliminate sub-optimal actions from our domain leading to a reduced connectivity and hopefully, smaller sizes of connected components.

Figure 1 shows the graphical representation of one simple MDP that has 7 states and 12 actions. In the figure, successors of probabilistic actions are connected by an arc. For simplicity reason, transition probabilities T_a and costs $C(s, a)$ are omitted. Using TVI, we can divide the MDP into two connected components $\mathcal{C}_1$ and $\mathcal{C}_2$. However, suppose we are given some additional information that a_5 and a_{12} are sub-optimal. Based on the remaining actions, $\mathcal{C}_1$ and $\mathcal{C}_2$ can be sub-divided into three and two smaller components respectively (as shown in the figure). Dynamic programming will greatly benefit from the new graphical structure,

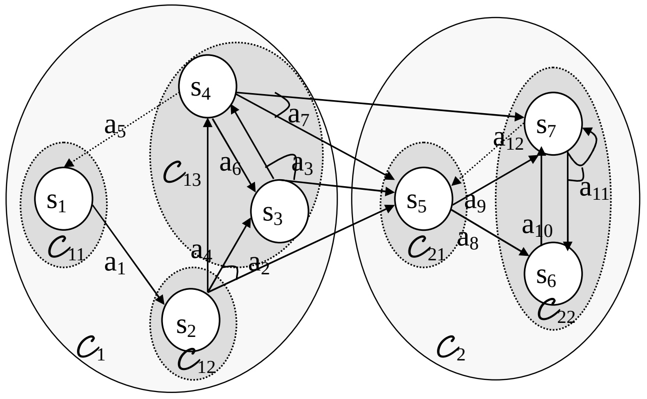

Figure 1: The graphical representation of an MDP and its set of strongly connected components (before and after the knowledge of some sub-optimal actions). Arcs represent probabilistic transitions, *e.g.*, a_7 has two probabilistic successors – s_5 and s_7.

since solving smaller components can be much easier than the larger ones.

The FTVI Algorithm

The key insight of our novel algorithm is to break the big components into smaller parts, by removing actions that can be proven suboptimal for the current problem at hand. This exploits the knowledge of the current initial state and goal, which TVI mostly ignores. We call our new algorithm focused topological value iteration (FTVI). The pseudo-code is shown in Algorithm 1.

At its core, FTVI makes use of the *action elimination* theorem, which states:

Theorem 1 Action Elimination *(Bertsekas 2000): If a lower bound of $Q^*(s, a)$ is greater than an upper bound of $V^*(s)$ then action a cannot be an optimal action for state s.*

This gives us a template to eliminate actions, except that we need to compute a lower bound for Q^* and an upper bound for V^*. FTVI keeps two bounds of V^* simultaneously: the lower bound $V_l(\cdot)$ and the upper bound $V_u(\cdot)$. $V_l(\cdot)$ is initialized via the admissible heuristic. We note two properties of V_l: (1) $Q(s, a)$ computed as in Line 29 of pseudo-code is a lower bound of $Q^*(s, a)$, and (2) all the values in value iteration remain a lower bound throughout the value iteration, if they were initialized by an admissible heuristic. So, this lets us easily compute a lower bound of Q^*, which also improves as more iterations are performed.

Similar properties hold for V_u, the upper bound of V^*, *i.e.*, if we initialize V_u by an upper bound and use backups over V_u then each successive value estimates remain upper bounds. Our implementation section below lists our exact procedure to compute the lower and upper bounds in a domain-independent manner. We note that to employ action elimination we can use any lower and upper bounds, so if a domain has informative, domain-dependent bounds available, that can be easily plugged into FTVI.

FTVI contains two sequential steps. In the first step, which we call the *search* step, FTVI performs a small number of heuristic searches similar to ILAO*, but differs fun-

Algorithm 1 Focused Topological Value Iteration

```
1: Input: an MDP $\langle \mathcal{S}, \mathcal{A}, T, C \rangle$, $x$: the number of search iterations in a batch, $y$: the lower bound of the percentage of change in the initial state value for a new batch of search iterations
2: // step 1: search
3: while true do
4:     $old\_value \leftarrow V_l(s_0)$
5:     for $iter \leftarrow 1$ to $x$ do
6:         for every state $s$ do
7:             mark every state as unvisited
8:         $s \leftarrow s_0$
9:         $Search(s)$
10:    if $old\_value/V_l(s_0) > (100 - y)\%$ then
11:        break
12:
13: // step 2: computation
14: build the graph $G_{SR}$
15: compute the strongly connected components of $G_{SR}$, order them by topological order $\mathcal{C}_1, \ldots, \mathcal{C}_k$
16: for $c \leftarrow 1$ to $k$ do
17:     solve component $\mathcal{C}_c$ by value iteration
18:
19: Function $Search(s)$
20: if $s \notin \mathcal{G}$ then
21:     mark $s$ as visited
22:     $a \leftarrow argmin_a Q(s,a)$
23:     for every unvisited successor $s'$ of action $a$ do
24:         $Search(s')$
25:     $Back-up(s)$
26:
27: Function $Back-up(s)$
28: for each action $a$ do
29:     $Q(s,a) \leftarrow C(s,a) + \sum_{s' \in \mathcal{S}} T_{a'}(s'|s) V_l(s')$
30:     if $Q(s,a) > V_u(s)$ then
31:         eliminate $a$ from $Ap(s)$
32: $V_l(s) \leftarrow min_{a \in Ap(s)} Q(s,a)$
33: $V_u(s) \leftarrow min_{a \in Ap(s)} [C(s,a) + \sum_{s' \in \mathcal{S}} T_{a'}(s'|s) V_u(s')]$
```

dermentally in that FTVI backs-up a state at most once per iteration. This difference makes the searches in FTVI much faster, but useful enough to eliminate sub-optimal actions. There are two other differences in common heuristic search and the search phase of FTVI. First, in each backup, we update the upper bound in the same manner as the lower bound. This is reminiscent of backups in Bounded RTDP (McMahan, Likhachev, and Gordon 2005). Second, we also check and eliminate sub-optimal actions using action elimination (Lines 28-33).

In the second step, the *computation* step, FTVI generates a directed graph G_{SR} in the same manner as TVI, but, only based on the remaining actions. More concretely, a directed edge from vertex s_1 to s_2 exists if there is an *uneliminated* action a such that $T_a(s_2|s_1) > 0$. It is easy to see that the graph G_{SR} generated is always a sub-graph of G_R – the one generated by TVI. FTVI then finds all connected components of G_{SR}, their topological order, and solves each component sequentially in the topological order.

We can state the following theorem for FTVI:

Theorem 2 *FTVI is guaranteed to converge to the optimal value function.*

The correctness of the theorem is based on two facts: 1) action elimination preserves soundness, and 2) TVI is an optimal planning algorithm (Dai and Goldsmith 2007).

Implementation

There are several interesting questions to answer in implementation. How to calculate the initial upper and lower bounds? How many search iterations do we need to perform in the search step? What if there still exists a big component even after action elimination?

We calculate our lower bound (admissible heuristic) as follows: We first simplify an MDP into a deterministic problem, splitting an action and its probabilistic transitions into several deterministic actions, with the same cost. We then solve this problem by a single backward systematic search from the set of goal states. Values of the deterministic problems are used as V_l.

We start with a simple upper bound:

$$\begin{aligned} V_u(s) &= 0 \quad \text{if } s \in \mathcal{G} \text{ else} \\ V_u(s) &= \infty \end{aligned} \tag{3}$$

This initialization gives us a global yet very imprecise upper bound. To improve its tightness, we perform a backward best-first search from the set of goal states. States visited have their V_u values updated as in Algorithm 1 Line 33. We can iteratively get tighter and tighter bounds when more backward searches are performed.

Amount of search can have a significant impact on FTVI. Very few search iterations might not eliminate enough sub-optimal actions. However, too many search iterations will turn FTVI into a heuristic search algorithm. Considering the tradeoff, we let the algorithm automatically determine the number of search iterations. FTVI incrementally performs a batch of x search iterations. After the batch, it computes the amount of change to the $V_l(s_0)$ value. If the change is over a percentage of y, a new batch of search is performed. Otherwise, the search phase is considered complete. In our implementation, we use $x = 100$, and $y = 3$.

Sometimes there are cases where G_{SR} still contains some large connected components. This can be caused by two reasons (1) an optimal policy indeed has large components, or (2) the connectivity caused by many suboptimal actions is not successfully eliminated by search. To try to further decompose these large components, we let FTVI perform additional *intra-component heuristic searches*. An intra-component heuristic search is a search that takes place only inside a particular component. Its aim is to find new sub-optimal actions, which might help decompose the component. Given a component $\mathcal{C}$ of G_{SR}, we define $Source_{\mathcal{C}}$ to be a set of states where none of its incoming transitions are from states in $\mathcal{C}$. In other words, states in $Source_{\mathcal{C}}$ are the incoming bridge states between $\mathcal{C}$ and rest of the MDP. An intra-component heuristic search of $\mathcal{C}$ originates from a state in $Source_{\mathcal{C}}$. A search branch terminates when a state outside $\mathcal{C}$ is encountered.

We did some experiments and compared the performance of FTVI with and without additional intra-component search on problems from four domains, namely Wet-floor (Bonet and Geffner 2006), Single-arm pendulum (Wingate and Seppi 2005), Drive and Elevator (ipc 2006). Our results show that additional intra-component search only provided limited gains in Wet-floor problems, in which it helped decrease the size of the largest components by approximately 50% on average, and sped up the convergence by 10% at best. However, intra-component search turned out to be harmful for the other domains, as it did not provide any new graphical information (no smaller components were generated). On the contrary, the search itself introduced a lot of unnecessary overhead. So we use the version that does not perform additional intra-component search throughout the rest of the experiments.

Experiments

We address the following two questions in our experiments: (1) How does FTVI compare with other algorithms on a broad range of domain problems? (2) What are the specific kind of domains on which FTVI should be preferred over heuristic search?

We used the fully optimized C code of ILAO* (Hansen and Zilberstein 2001). We additionally implemented LRTDP (Bonet and Geffner 2003b), BRTDP (McMahan, Likhachev, and Gordon 2005), HDP (Bonet and Geffner 2003a), TVI, and FTVI over the same framework. We performed all experiments on a 2.5GHz Dual-Core AMD Opteron(tm) Processor with 2GB memory. We used a cutoff time of 5 minutes for each algorithm per problem. We used a threshold value $\epsilon = 10^{-6}$. We ran BRTDP on the same upper bound as FTVI, and used $\alpha = 2 \times 10^{-6}$ and $\tau = 10$. We found HDP too slow, so do not report its performance.

Relative Speed of FTVI

We evaluate the various algorithms on problems from seven domains – Mountain Car, Single and Double Arm Pendulum (Wingate and Seppi 2005), Wet-floor (Bonet and Geffner 2006)[2], and three domains from International Planning Competition 2006 – Drive, Elevators and TireWorld. A mountain car problem usually has many source states.[3] We choose each source state as an initial state, and average the statistics per problem. Table 1 lists the running times for the various algorithms ($Time$). For FTVI, we additionally report the time used by the searches (T_{search}), and the time spent in generating the graphical structure (T_{gen}), where the leftover is the time spent in solving the SCCs. We also compare the size of the biggest component (BC size) generated by TVI and FTVI.

Overall we find that FTVI outperforms the other four algorithms on most of these domains. FTVI outperforms TVI in all but the DAP domain (in DAP FTVI's search step takes too long). Notice that on MCar and Tireworld problems, FTVI establishes very favorable graphical structures (strongly connected components of size one) during the search step. This graphical structure makes the second step of FTVI trivial. But, TVI has to solve much bigger components, so it runs much slower. FTVI outperforms heuristic search algorithms most significantly in domains such as MCar, SAP and Drive. It is faster than ILAO* by an order of magnitude. The two RTDP algorithms are not competitive with the other algorithms in these domains, and fail to return a solution by the cutoff time for many problems. In contrast, FTVI shows limited speedup against heuristic search in domains such as Wet-floor, DAP, and Elevator. FTVI is slower than heuristic search in Tireworld problems, largely due to its overhead in constructing the graphical structures.

Factors Determining Performance

We have shown that FTVI is faster than heuristic search in many domains, but its relative speedup is somewhat domain-dependent. Can we find any domain features that are particularly beneficial for FTVI or worse for heuristic search? If so, then we can make an informed choice regarding the best MDP solver to use. In this evaluation we perform control experiments by varying the domains across different features and study the effect on planning time of various algorithms.

We make an initial prediction of three features:

1. The number of goals in the domain: If the number of goal states is small search may take a long time before it discovers a path to a goal. Therefore, many sub-optimal policies might be evaluated by a heuristic search algorithm.
2. Search depth from the initial state to a goal state: This depth is a lower bound of the length of an execution trial and also of the *size* of any policy: the number of reachable states under that policy. A longer depth implies more search iterations, which might make evaluating a policy (in the form of probabilistic paths) time-consuming.
3. Heuristic informativeness: The performance of a heuristic-search algorithm depends a lot on the quality of the initial heuristic function. We expect the win from FTVI to increase when heuristic is less informed.

The Number of Goals As far as we know, there is no suitable domain where we can specify the total number of goal states arbitrarily, so we use a randomly-generated, artificial domain. In this domain each state has two applicable actions, and each action has at most two random successors. We test all algorithms on domains of two sizes, 10,000 (Figure 2(a)) and 100,000 (Figure 2(b)). For each problem size, we vary the number of goal states $|\mathcal{G}|$. For each $|\mathcal{G}|$ value, we generate 10 problems, and report the median running time of four algorithms (LRTDP was slow in this domain). We observe that all algorithms take more time to solve a problem with a smaller number of goal states than with a larger number. However, beyond a point ($|\mathcal{G}| > 20$ in our experiments), the running times become stable. FTVI runs only marginally slower when $|\mathcal{G}|$ is small, suggesting that its performance is relatively less dependent on the number of goal states. BRTDP is the second best in handling small number of goal states, and it runs faster than FTVI when the number of goal states is large.

Search Depth In this experiment, we study how the search depth of a goal from the initial state influences the perfor-

[2]Note that we used the probability of wet cells, $p = 0.5$.

[3]A source state is a state with no incoming transitions.

Problem	Reachable States	ILAO*	LRTDP	BRTDP	TVI		FTVI			
		Time	*Time*	*Time*	BC size	*Time*	BC size	T_{search}	T_{gen}	*Time*
MCar100	10,000	1.91	1.23	2.81	7,799	0.68	1	0.20	0.01	**0.22**
MCar300	90,000	11.91	229.70	117.23	71,751	23.22	1	2.22	0.13	**2.35**
MCar700	490,000	101.65	-	216.01	390,191	233.98	1	12.29	0.76	**13.06**
SAP100	10,000	1.81	2.58	9.39	9,999	2.37	510	0.15	0.01	**0.17**
SAP300	90,000	32.4	-	-	89,999	44.2	43,251	2.63	0.20	**5.99**
SAP500	250,000	130.17	-	-	-	-	121,143	8.53	0.55	**24.20**
WF200	40,000	11.22	-	22.08	39,999	20.58	15,039	3.30	0.12	**8.81**
WF400	160,000	98.97	-	97.73	159,999	100.78	141,671	14.27	0.36	**74.24**
DAP10	10,000	1.02	51.45	3.04	9,454	**0.75**	9,250	0.82	0.04	0.89
DAP20	160,000	32.68	-	144.12	150,489	**21.95**	146,159	22.33	0.74	23.98
Drive	4,563	1.60	**0.69**	7.85	4,560	1.23	4,560	0.01	0.02	1.07
Drive	29,403	96.09	273.37	163.91	29400	13.03	29,400	0.02	0.15	**10.63**
Drive	75,840	-	-	-	75,840	74.70	75,840	0.03	0.40	**41.93**
Elevator (IPPC p13)	539,136	227.53	-	-	1053	58.46	1,053	0.01	1.73	**54.11**
Elevator (IPPC p15)	539,136	27.35	-	-	1053	14.59	1,053	0.01	1.60	**12.11**
Tireworld (IPPC p5)	671,687	**0.01**	0.14	**0.01**	23	2.26	1	0.00	1.26	1.26
Tireworld (IPPC p6)	724,933	**0.01**	0.16	**0.01**	618,448	48.81	1	0.00	1.44	1.44

Table 1: Performance of the different algorithms on problems in various domains. FTVI outperforms all algorithms by vast margins. (BC size means the size of the biggest connected component. All running times are in seconds. Fastest times are bolded. '-' means that the algorithm failed to solve the problem within 5 minutes.)

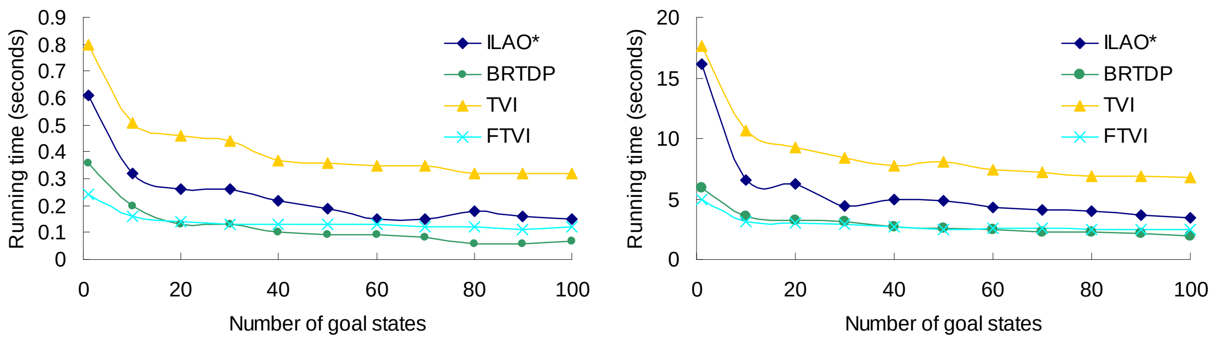

Figure 2: Running time of algorithms with different number of goal states and problem size (left) $|S| = 10,000$ (right) $|S| = 100,000$ in random MDPs. All algorithms except FTVI slow down massively when the number of goal states is small.

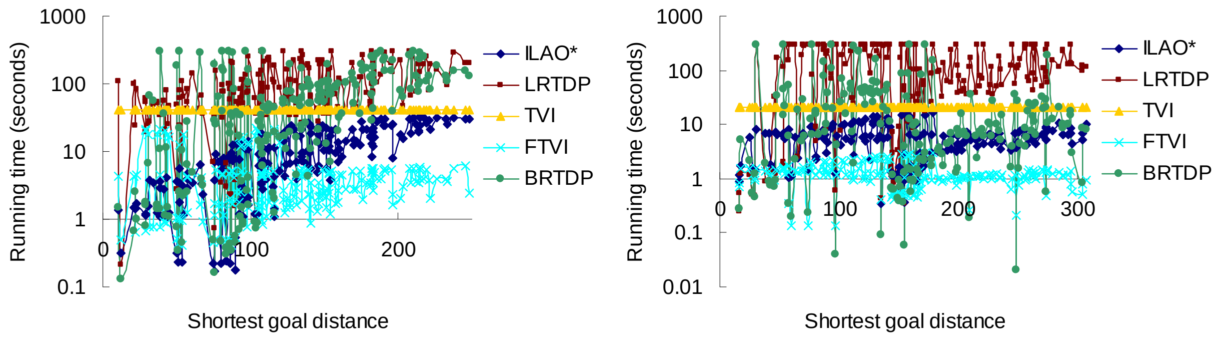

Figure 3: Running time of algorithms with different shortest distance to the goal on (left) mountain car 300×300 (MCar300) (right) single-arm pendulum 300×300 (SAP300) problems. Heuristic search algorithms slow down massively when the number search depth is large.

mance of various algorithms. We choose a Mountain car problem and a Single-arm pendulum problem. We randomly pick initial states from the state space[4], measure the search depth, *i.e.*, the shortest distance to a goal state, d. The running times in Figure 3 are ordered by d.

As we can see from Figure 3, TVI's performance is unaffected by the search depth, which is expected, since it is a variant of value iteration and has no search component. LRTDP runs the fastest when d is relatively small, but it slows down considerably and is unable to solve many problems when d becomes larger. ILAO*'s convergence speed varies a bit when the distance is small. As d increases, its running time also increases. BRTDP's performance is very close to that of ILAO*, except that it is slower in SAP300 when d is large. ILAO*, LRTDP and BRTDP suffer the most from the increase in the search depth, since they are pure search algorithms. FTVI, the fastest algorithm in this suite of experiments, converges very quickly for all initial states (usually around one or two seconds on Mcar300, and less than 10 seconds on SAP300). We observe that as d increases, FTVI does not slow down and it converges one order of magnitude faster than ILAO*, BRTDP and TVI, and two orders of magnitude faster than LRTDP for large depths.

Heuristic Quality Finally we study the effect of the heuristic informativeness on the algorithms. We conduct two sets of experiments, based on two sets of consistent heuristics. We find BRTDP slower than other algorithms in these experiments, so do not plot its running times in the figures. In the first experiment, we pre-compute the optimal value function of a problem using value iteration, and use a fraction of the optimal value as an initial heuristic. Given a fraction f, we calculate $h(s) = f \times V^*(s)$. Figure 4 plots the running time of different algorithms against f for three problems. Note that $f = 1$ means the initial heuristic is already optimal, so a problem is trivial for all algorithms, but TVI and FTVI have the overhead of building a topological structure. LRTDP is slow in the Wet100 problem, so its running times in the Wet100 problem are omitted from the figure. The figure shows that as f increases (i.e. as the heuristic becomes more informative) the running times of all algorithms decrease almost linearly. This is true even for TVI, which is not a heuristically-guided algorithm, but takes less time probably because the initial values affect the number of iterations required till convergence.

To thoroughly study the influence of the heuristics, we conduct a second set of experiments. In this experiment, we use a fractional V_l value as our initial heuristic. Recall that V_l is a lower bound of V^* computed by the value of a determinized problem. We calculate the initial heuristic by $h(s) = f \times V_l(s)$. All included algorithms show a similar smooth decrease in running time when f increases. BRTDP, however, shows strong dependence on the heuristics in the Wet100 problem. Its running time decreases sharply from 120.45 seconds to 0.6 seconds and from 124.36 seconds to 7.54 seconds from when $f = 0.02$ to when $f = 1$ in the two experiments. Stable changes in the two experiments suggests the following for algorithms except BRTDP: (1) No algorithm is particularly vulnerable to a less informed heuristic function; (2) extremely informative heuristics (when f is very close to 1) do not necessarily lead to extra fast convergence. This result is in line with recent results in (Helmert and Röger 2008) for deterministic domains.

[4]Note that these problems have well-defined initial states. Here we pick initial states arbitrarily from $\mathcal{S}$.

Discussion

From the experiments, we learn that FTVI is vastly better in domains whose problems have a small number of goal states and a long search depth from the initial state to a goal (such as MCar, SAP and Drive). In contrast, heuristic search performs the best for domains whose problems have many goal states and the search depth is short (such as Tireworld). In addition, FTVI displays limited advantage over heuristic search in the two intermediate cases where a problem has (1) many goal states but long search depth (Elevator), (2) a short depth but fewer goal states (DAP). In conclusion, FTVI is our algorithm of choice whenever a problem has either a small number of goal states or a long search depth.

Related Work

Besides TVI several other researchers have proposed decomposing an MDP into subproblems and combining their solutions for the final policy, *e.g.*, (Hauskrecht et al. 1998; Parr 1998). However, these approaches typically assume some additional structure in the problem, either known hierarchies, or known decomposition into weakly coupled sub-MDPs, *etc.*, whereas FTVI assumes no additional structure. Moreover, FTVI is optimal whereas other algorithms are approximate.

BRTDP also keeps an upper bound for the value function. But, it uses the upper bound purely to judge how close a state is to convergence, by comparing the difference between the upper and lower bound values. The algorithm tries to make searches focus more on states whose two bounds have larger differences, or intuitively, states whose values are less converged. Unlike FTVI, BRTDP does not perform action elimination, nor does it use any connected component information to solve an MDP. Furthermore, unlike FTVI its performance is highly dependent on the quality of the heuristics.

Conclusions

This paper makes several contributions. First, we present a new optimal algorithm to solve MDPs, focused topological value iteration (FTVI), which extends topological value iteration algorithm by focusing the construction of strongly connected components on transitions that likely belong to an optimal policy. FTVI does this by using a small amount of heuristic search to eliminate provably suboptimal actions. In contrast to TVI, which does not care about goal-state information, FTVI ignores transitions which it determines to be irrelevant to an optimal policy for reaching the goal. In this sense, FTVI builds a much more informative topological structure than TVI.

Second, we show empirically that FTVI outperforms TVI in many domains, usually by an order of magnitude. This

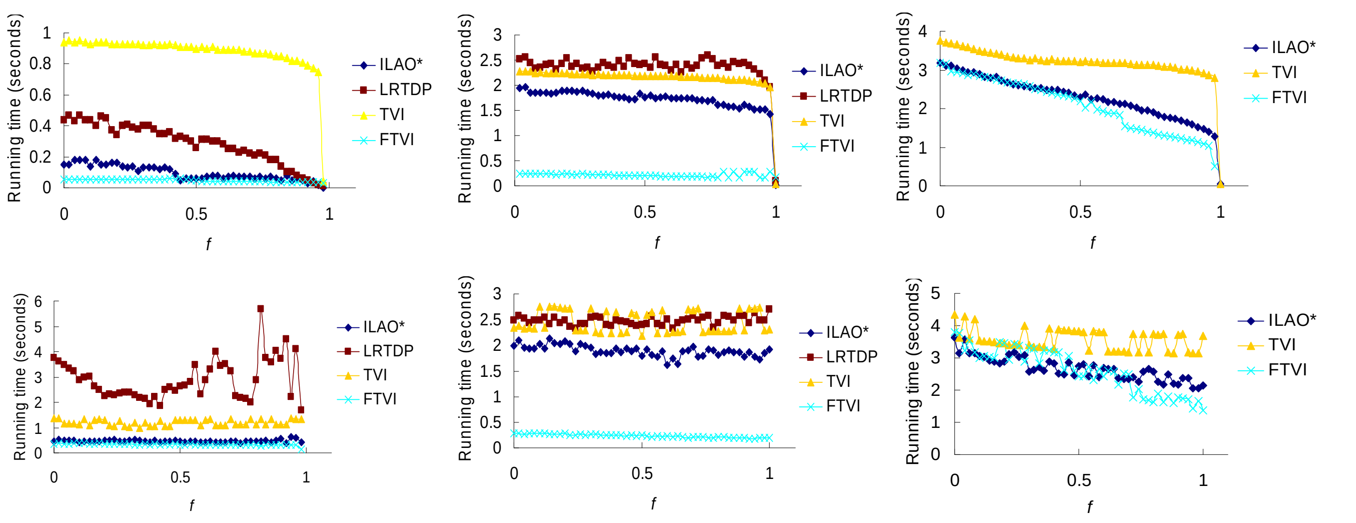

Figure 4: Running time of algorithms with different initial heuristic on (left) mountain car 100×100 (MCar100) (middle) single-arm pendulum 100×100 (SAP100) (right) wet-floor 100×100 (WF100) problems. All algorithms are equally sensitive to the heuristic informativeness. (top) $f = \sum_{s \in \mathcal{S}} h(s) / \sum_{s \in \mathcal{S}} V^*(s)$ (bottom) $f = \sum_{s \in \mathcal{S}} h(s) / \sum_{s \in \mathcal{S}} V_l(s)$.

performance is due to the success of a more informed graphical structure, since the size of the connected components found by FTVI are vastly smaller than those constructed by TVI's.

Third, we find that for many domains FTVI massively outperforms popular heuristic search algorithms, such as ILAO* and LRTDP. After analyzing the performance of these algorithms over different problems, we find that a smaller number of goal states and long search depth to a goal are two key features of problems that are especially hard for heuristic search to handle. Our results show that FTVI outperforms heuristic search in such domains by an order of magnitude.

Acknowledgments

This work was supported by Office of Naval Research grant N00014-06-1-0147 and the WRF / TJ Cable Professorship. We thank Eric A. Hansen for sharing his code for ILAO*, and anonymous reviewers for excellent suggestions on improving this paper.

References

Aberdeen, D.; Thiébaux, S.; and Zhang, L. 2004. Decision-theoretic military operations planning. In *ICAPS*, 402–412.

Barto, A.; Bradtke, S.; and Singh, S. 1995. Learning to act using real-time dynamic programming. *AI J.* 72:81–138.

Bellman, R. 1957. *Dynamic Programming*. Princeton University Press.

Bertsekas, D. P., and Tsitsiklis, J. N. 1996. *Neuro-Dynamic Programming*. Athena Scientific.

Bertsekas, D. P. 2000. *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific.

Bonet, B., and Geffner, H. 2003a. Faster heuristic search algorithms for planning with uncertainty and full feedback. In *IJCAI*, 1233–1238. Morgan Kaufmann.

Bonet, B., and Geffner, H. 2003b. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *ICAPS*, 12–21.

Bonet, B., and Geffner, H. 2006. Learning in depth-first search: A unified approach to heuristic search in deterministic non-deterministic settings, and its applications to MDPs. In *ICAPS*, 142–151.

Bresina, J. L.; Dearden, R.; Meuleau, N.; Ramkrishnan, S.; Smith, D. E.; and Washington, R. 2002. Planning under continuous time and resource uncertainty: A challenge for AI. In *UAI*, 77–84.

Dai, P., and Goldsmith, J. 2007. Topological value iteration algorithm for Markov decision processes. In *IJCAI*, 1860–1865.

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

Hauskrecht, M.; Meuleau, N.; Kaelbling, L. P.; Dean, T.; and Boutilier, C. 1998. Hierarchical solution of Markov decision processes using macro-actions. In *UAI*, 220–229.

Helmert, M., and Röger, G. 2008. How good is almost perfect? In *AAAI*, 944–949.

2006. http://www.ldc.usb.ve/ bonet/ipc5/.

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

McMahan, H. B.; Likhachev, M.; and Gordon, G. J. 2005. Bounded real-time dynamic programming: Rtdp with monotone upper bounds and performance guarantees. In *ICML*, 569–576.

Musliner, D. J.; Carciofini, J.; Goldman, R. P.; E. H. Durfee, J. W.; and Boddy, M. S. 2007. Flexibly integrating deliberation and execution in decision-theoretic agents. In *ICAPS Workshop on Planning and Plan-Execution for Real-World Systems*.

Parr, R. 1998. Flexible decomposition algorithms for weakly coupled Markov decision problems. In *UAI*, 422–430.

Smith, T., and Simmons, R. G. 2006. Focused real-time dynamic programming for mdps: Squeezing more out of a heuristic. In *AAAI*.

Wingate, D., and Seppi, K. D. 2005. Prioritization methods for accelerating MDP solvers. *JMLR* 6:851–881.