A Novel Prioritization Technique for Solving Markov Decision Processes

Jilles S. Dibangoye
Greyc-Cnrs & DAMAS
Université Laval, Q.C., Canada
gdibango@info.unicaen.fr

Brahim Chaib-draa
DAMAS Laboratory
Université Laval, Q.C., Canada
chaib@ift.ulaval.ca

Abdel-illah Mouaddib
Greyc-Cnrs
Université de Caen, France
mouaddib@info.unicaen.fr

Abstract

We address the problem of computing an optimal value function for Markov decision processes. Since finding this function quickly and accurately requires substantial computation effort, techniques that accelerate fundamental algorithms have been a main focus of research. Among them prioritization solvers suggest solutions to the problem of ordering backup operations. Prioritization techniques for ordering the sequence of backup operations reduce the number of needed backups considerably, but involve significant overhead. This paper provides a new way to order backups, based on a mapping of states space into a metric space. Empirical evaluation verifies that our method achieves the best balance between the number of backups executed and the effort required to prioritized backups, showing order of magnitude improvement in runtime over number of benchmarks.

Introduction

Markov decision processes (MDPs) have proven very useful to model a variety of problems in sequential decision-making by individual agents. However, the computational difficulty of applying fundamental algorithms, *e.g.,* value iteration (VI) and policy iteration (PI), has spurred much research into methods that accelerate fundamental solvers by means of states aggregation or abstraction; decomposition; and more recently prioritization.

Aggregation and abstraction techniques perform domains encoding through a compact representation using various model equivalence criteria, *e.g.,* bisimulation (Givan, Dean, & Greig 2003). Unfortunately, the search space for optimal policies grows exponentially with the size of any compact encoding. A common way to cope with this issue relies on decomposition techniques (Dean & Lin 1995). Nevertheless, these approaches are proven to be efficient only on weakly coupled problems (Parr 1998). More promising methods rely on hybrid techniques that complete in the same computational mechanism both states abstraction and decomposition (Hoey *et al.* 1999). While interesting and useful none of the above approaches have direct bearing on our approach, and thus we will not pursue them further here.

VI and PI are inefficient mainly because they proceed by *backing up* the entire state space at each iteration even if unnecessary or redundant and computationally demanding. In the remainder of this paper *backup* refers to Bellman update or *Bellman backup*, that is, the way the value of a state is updated. It has been proven that by backing up states in the right order, one can improve the performance of MDP solution methods (Wingate & Seppi 2005). In contrast to fundamental MDP solvers prioritization techniques attempt to compute good sequence of backups. Prioritization techniques for ordering the sequence of backup operations reduce the number of needed backups considerably, but can involve excessive overhead. Indeed none of prioritization solvers use a backup order that is built only once, *i.e.,* stationary backup order, rather they all proceed by dynamically updating state priorities as discussed later. One exception is topological value iteration (TVI) (Dai & Goldsmith 2007). TVI suggests performing backups in the order of a causal relation among layers of states. Although TVI is a great improvement on previous methods, it suffers from two major drawbacks: first, as already mentioned computing the backup order incurs non-negligible overhead; moreover when facing domains where all states are mutually causally related TVI is no better than value iteration and thus inherits its value iteration's disadvantages. These are strong arguments that outline the necessity of improving state of the art TVI algorithm. While this algorithm fails to scale to general MDPs, it shows promising results demonstrating the potential of prioritization methods in solving MDPs.

The main contribution of this paper is an improved version of topological value iteration (iTVI). This algorithm is based on the insight that given any MDP, one can always find a mapping of states space S into a metric space (S, d); hence one can perform backups in the order of metric d. We determine this metric by means of generalizing TVI's causal relation among layers to a causal relation among states. iTVI uses standard shortest-path techniques, *e.g.,* Dijkstra, to quantitatively measure d. Doing so permits us to prioritize the backups of states in a single layer in contrast to TVI algorithm thus greatly increasing TVI's applicability to general MDPs. When dealing with techniques that attempt to accelerate fundamental algorithms it is always questionable whether the effort required to perform offline steps, *e.g.,* abstraction, decomposition, prioritization or computation of heuristic values, is not prohibitively high in relation to gains expected. Following this observation, we suggest a new way

to order backups, based on metric space (S, d) that achieves the best balance between the number of backups executed and the effort required to prioritize backups. The experiments confirm that our algorithm scales up better than TVI on different MDP models.

The rest of the paper is structured as follows: we cover first the basic model for stochastic problems, as well as the state of the art solution methods. We then describe domains that motivate this work. In Section 4, we introduce improved topological value iteration algorithm. We finally run an empirical evaluation of the resulting solver and end with a brief discussion in Section 5.

Background and Related Work

In this Section, we provide a brief review of Markov decision processes and some of the state of the art techniques.

Markov decision processes

An MDP M is a tuple (S, A, P, R, γ) where: S is a discrete and finite state space; A is a discrete and finite action space; $P(s'|s, a)$ is a function of transition probabilities, *i.e.,* the probability of transiting from state s to state s' when taking action a; $R(s, a)$ is a real-valued reward function, that defines the outcomes received when taking action a in state s; and γ is a discount factor.

Given an MDP model, the goal is to find a function $\pi\colon S \to A$ mapping states s into action a. Such a function is called a policy. An optimal policy is a policy $\pi^\star$ that maximizes (resp. minimizes) the expected long-run reward:

$$\pi^\star(s) = \arg\max_{a\in A} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid s_0 = s\right] \qquad (1)$$

A policy π can be computed using its expected value denoted V^π by choosing an action for each state that contributes its value function. The optimal value function $V^{\pi^\star}$ is determined by repeated application of the so called Bellman equation:

$$V(s) = \max_{a\in A} \left[R(s, a) + \gamma \sum_{s'\in S} P(s'|s, a)V(s')\right] \qquad (2)$$

until some stopping criterion is reached.

One fundamental algorithm for solving MDPs is value iteration. In *synchronous* value iteration all states are updated in parallel resulting in considerable useless and redundant backups. Rather in *asynchronous* value iteration algorithms, at each iteration, only a small subset of states is selected for update as discussed in the following.

Asynchronous methods

The first asynchronous approaches credited concurrently to (Sutton 1991) and (Peng & Williams 1993), suggest that we backup the values of states in a random order. More recent asynchronous algorithms rely on prioritized sweeping (PS) methods (Sutton & Barto 1998). In such a method, a queue of every state-action pairs whose estimated value would change nontrivially if backed up, is maintained prioritized by the size of the change. When the top pair in the queue is backed up, the effect on each of its predecessor pairs is computed. If the effect is greater than a certain small threshold, then the pair is inserted in the queue with the new priority (McMahan & Gordon 2005; Wingate & Seppi 2005). Although prioritized sweeping methods are a great improvement on previous methods, the overhead of managing the priority queue can be prohibitively high.

The idea of computing good backup sequences has also been used in heuristic techniques for efficiently solving special types of MDPs. For MDPs with positive cost and a few start states, forward search algorithms, *e.g.,* LRTDP (Bonet & Geffner 2003a), LAO* (Hansen & Zilberstein 2001) and HDP (Bonet & Geffner 2003b), are the state of the art; and for MDPs with few goal states, backward search algorithms, *e.g.,* improved PS (McMahan & Gordon 2005), have shown interesting results. As already mentioned, finding good backup orders typically incurs considerable computational overhead; hence, even though the number of backups is reduced, in most such schemes the benefits do not fully manifest in total execution time. This has led to the development of prioritized techniques that make use of *stationary* backup orders instead of dynamically adjusted ones as previously required.

Topological value iteration

Topological value iteration has been introduced by (Dai & Goldsmith 2007). TVI differs from previous prioritized algorithms since the overhead of computing its stationary backup order is only linear in the size of the MDP graphical representation[1] denoted G. TVI proceeds first by grouping states that are mutually causally related together and then by making them as layers of states denoted $\{S^\ell\}_\ell$. The problem of finding layers of an MDP M corresponds to the problem of finding the strongly connected components of its graphical representation G. Then, TVI lets these layers form a new MDP called reduced MDP and denoted M'. In this case, the reversed topological sorting of layers provides the so called topological backup order ($\prec_1$). Finally, TVI proceeds by backing up layers $\{S^\ell\}_\ell$ in only one *virtual* iteration in the order of its topological backup order.

Nevertheless, the causal relation used in TVI does not fully leverage the advantage sparse domain dynamics may offer. Indeed, while TVI demonstrates good performances on layered MDPs, it suffers from a number of drawbacks in general MDPs. In particular, when all states are in a single layer TVI is no better than value iteration. This is mainly because, the topological backup order used in TVI is unable to prioritize the order in which states in a single layer S^ℓ should be backed up. The problem with the causal relation used in TVI is that it is based only on the mutual dependency among states. More precisely, it does not distinguish between different levels of dependency among states. At the moment TVI processes a single layer, it looks at solving it as if the level of dependency among states in that layer is the same. So the topological backup order used in TVI does not

[1]The graphical representation of an MDP M is a directed graph $G(S, E)$. The set S has vertices where each vertex represents a state $s \in S$. The edges, E, in G represent transitions from one state s to another s' ($s \rightsquigarrow s'$), *i.e.,* there exists an action $a \in A$ such that taking action a in state s may lead into state s'.

leverage the advantage sparse layers may offer. With this intuition we want to design a backup order that is able to fully exploit the sparsity of domains while being computationally less expensive.

Motivating examples

In this Section, we describe problem examples and their structural properties that originally motivate this work.

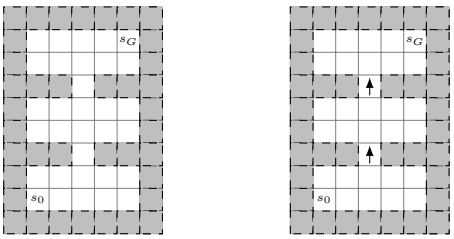

Figure 1: Two examples of #.maze domains.

Race track problem

The Race Track is a planning problem that simulates automobile racing (Barto, Bradtke, & Singh 1993). The modified version assumes only a probabilistic single car evolving in a road racing represented by a shape (see Figure **??**) and composed with a starting and a finishing line and many weakly-coupled roads. For the sake of simplicity, we assume only 4 straightforward moves (right, left, up, and down). In contrast with the original race track domain, we allow the obstacle states to be dead ends, that is when a car falls into such states there is no action leading outside. The shape can be layered such as small.maze or only weakly coupled as depicted in small-bi.maze Figure 1.

Stochastic traveling salesman problem

Some planning and scheduling problems are such that the overall goal of the problem is to complete some set of subtasks (see Figure 2). In this case the problem will typically have dimensionality that is at least linear in the number of subtasks. To look at such applications, we consider the stochastic traveling salesman problem (STSP) defined as follows. Let M be a communicating MDP[2], $C \subseteq S$ be the set of subtask states that must be visited, and let $s_0 \in C$ be a designated start state. Then, the stochastic traveling salesman problem (STSP) on $\langle M, C, s_0 \rangle$ is to find a policy (most likely non-stationary, history dependent) that minimizes the expected time to visit all $s \in C$ at least once, starting from s_0 and returning to s_0 after all states in C are visited. While STSP is a non-stationary application, to fit within the MDP framework, we create an augmented MDP $\hat{M}$ that extends the state space of M with $|C|$ additional boolean variables which indicate which states in C have been visited. Then, we make the goal in $\hat{M}$ to have all $|C|$ additional variables $true$ while in s_0, in such a way that optimal solutions to $\hat{M}$ correspond directly to solutions of the STSP.

[2] An MDP is communicating if all states are in a single layer.

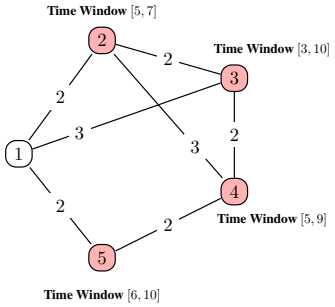

Figure 2: Illustration of STSP domain with time windows.

Structural properties

These two domains embody the graphical features TVI and its extension iTVI try to leverage. The domain illustrated by small.maze is said to be layered since some states (cells with an arrow) can only perform a single deterministic action (direction of the arrow). Once the system reaches this state, it enters a new layer and it cannot go back; hence the domain can be partitioned into multiple layers. Similarly with regard to small.maze, in STSP when a subtask state $s \in C$ is visited the associated boolean variable v_s is set to $true$. Then, the system will never fall into states of the augmented MDP where v_s is set to $false$ (since s is already visited). Therefore both STSP and small.maze are layered MDPs. However, there are a number of applications, as for instance small-bi.maze, that yield sparse domain dynamics but are not layered (when considering only non dead-end states). Such domains are known in the MDP literature as weakly (or loosely) coupled MDPs and match a variety of realistic applications (Parr 1998). Weakly coupled MDPs are domains where one can find state subspaces whose interSections have at least $k > 0$ vertices. Unfortunately, the problem of finding weakly coupled subspaces of an MDP corresponds to the problem of finding k-connected components of its graphical representation G. The complexity of such decomposition algorithms is bounded in the worst-case by $O(|S^{\ell}|^{k})$. Hence, computing then ordering such state subspaces is intractable in general.

Improved TVI

The purpose of iTVI is to provide a stationary backup order that prioritizes states rather than state subspaces. To overcome the difficulties in computing good backup orders, mostly due to the excessive overhead, we draw our inspiration from (Dai & Goldsmith 2007)'s observation: 'states and their value functions are causally related'. Pushing this observation further, we note that states and their value functions have different levels of dependency. In an MDP M, one state s'' is a successor of state s' which is the successor of state s, *i.e.,* $s \rightsquigarrow s' \rightsquigarrow s''$. Then, $V(s)$ is dependent on both $V(s')$ and $V(s'')$. However, $V(s)$ is much more dependent on $V(s')$ than $V(s'')$, that is the level of dependency between s and s' is greater than the one between s and s''. Indeed, the effect of $V(s')$ on $V(s)$ arises only after a single sweep of the value function, while the effect on

$V(s)$ of $V(s'')$ requires two sweeps. For these reasons, we want to backup s'' ahead of s' and s' ahead of s. This causal relation is transitive. Unfortunately, such a relation is common among states since MDPs are cyclic. This suggests that a potential way for finding a good sequence of backups is to group states for different levels of dependency. Nevertheless, the problem of finding such clusters of states corresponds to the problem of finding weakly coupled states subspaces, and as discussed above this problem is intractable. Here, we take a different approach that has interesting consequences both from practical and theoretical point of view.

For the sake of simplicity, we assume given a single start state s_0 although the following exposition can be easily extended to cope with multiple start states. Roughly, we keep track of a metric $d(s)$ over each reachable state s — where $d(s)$ denotes the minimum number of sweeps of the value function that is required before $V(s_0)$ benefits from $V(s)$. Given the graphical representation G of an MDP, the problem of determining the minimum number of sweeps is equivalent to the single-source shortest path problem, where the graph is G and each arc is labeled with $(+1)$ and the source vertex is s_0. For solving such problems one can use Drijkstra's algorithm that runs in time $O(|S|\log|S|)$. Here, we make use of a more accurate method as explained below.

To build our backup order we proceed by means of a twofold method : (1) we perform a depth-first-search algorithm that collects all states reachable from start state s_0; (2) we apply a breadth first search algorithm (BFS) to build our metric d (see Algorithm 1). Indeed, the first-in-first-out (FIFO) order in which BFS processes vertices in $\bar{G}$ is the reverse of the backup order we are looking for. The basic idea is to perform a BFS in the *transpose* of $\bar{G}$. The search starts at state s_0 taken as the root vertex. Then, states are added in the queue as they are discovered and processed in FIFO order. The search algorithm namely BackUpOrder labels each processed state s by $d(s)$, the distance (metric) from the root to s. Hence, states can be partitioned into levels based on the metric $d(s)$. Furthermore, the algorithm maintains values $V(s)$ such that possible ties of distance d could be broken. This provides the construction steps of functions d and V. Given metric space (S, d), it is easy to build a backup order as follows: Consider the partial order $\prec_2$ over states S such that for all pairs $s, s' \in S$, s' is backed up ahead of s if s is closer to start state s_0 than state s', more formally: $\forall s, s' \in S\colon\ d(s) < d(s') \Rightarrow s \prec_2 s'$. Unfortunately, the metric space (S, d) fails to differentiate between states $s, s' \in S$ that yields the same metric value, *i.e.,* $d(s) = d(s')$. We use a heuristic function V to resolve ties by means of backing up s' and s in the order of V: if the goal is to maximize the expected long-run reward s' is backed up ahead of s when the heuristic value of s' is bigger or equal to the heuristic value of s and *vice-versa*, more formally: $\forall s, s' \in S\colon\ (d(s) = d(s') \wedge V(s) \leq V(s')) \Rightarrow s \prec_2 s'$. Obviously $\prec_2$ is only a partial order since we do not resolve ties of the heuristic value. Moreover algorithm BackUpOrder runs in time $O(|A||S|^2)$ that corresponds to the complexity of a single sweep of the value function.

The pseudocode of iTVI is shown in Algorithm 1. The algorithm proceeds first by computing its stationary backup order, then it iteratively updates the values of *reachable* states in the order of $\prec_2$. iTVI can be applied to MDPs with an arbitrary number of goal states, but to simplify the exposition, the description in Algorithm 1 is for MDPs with a start state s_0 and a single goal state s_G.

Algorithm 1: Improved TVI.

```
iTVI(M, ε)
begin
    BackUpOrder(V, d, G, M)
    Initialize backup order ≺2, and reachable states S̄.
    repeat
        Cache the current value function V' ← V.
        forall s ∈ S̄ in order of ≺2 do
            V(s) ←
            max_{a∈A} [R(s,a) + γ Σ_{s'∈S} P(s'|s,a)V(s')]
    until ||V − V'||_∞ ≤ ε
end

BackUpOrder(V, d, G, M)
begin
    MakeEmptyQueue(Q).
    Ḡ ← DFS(G, s_0, s_G)
    Ḡ ← Ḡ^⊤
    forall vertices s in Ḡ do
        visited(s) ← false.
        d(s) ← ∞, V(s) ← nil.
        d(s_0) ← nil.
        V(s_0) ← max_{a∈A} R(s_0, a).
        Enqueue(Q, s_0).
        visited(s_0) ← true.
    repeat
        s' ← Dequeue(Q).
        V(s') ←
        max_{a∈A} [R(s',a) + γ Σ_{s∈S} P(s|s',a)V(s)]
        forall vertices s ∈ Ḡ.adj[s'] do
            if ¬visited(s) then
                visited(s) ← true.
                d(s) ← d(s') + 1.
                Enqueue(s, Q).
    until Empty(Q)
end
```

Theorem 1 *iTVI is guaranteed to converge to the optimal value function over all reachable states.*

iTVI is a simple value iteration algorithm that backups all reachable states iteratively exactly like VI. Because planning over unreachable states is unnecessary, the convergence property and the optimality of VI guarantee the optimality of iTVI over all reachable states.

Empirical Evaluation

To evaluate iTVI, we compare it for different MDP models to other state of the art MDP solvers namely, Gauß-Seidel VI, TVI (Dai & Goldsmith 2007), PS (Wingate & Seppi

2005) as well as RTDP-style methods (for MDPs with positive costs). We introduce an additional algorithm namely iTVI$^+$, an improved version of iTVI that uses in addition layers $\{S^\ell\}_\ell$ provided by TVI. iTVI$^+$ updates each layer in the order of TVI's backup order $\prec_1$ and states in each layer in the order of iTVI backup order $\prec_2$. All algorithms are run over various maze domains with different MDP models as illustrated in Table 1. Indeed, to provide a more complete test-bed for all algorithms including specialized solvers, we allow different MDP models from the general form (no assumption on the MDP model) to those including additional assumptions, *e.g.,* for MDPs with positive cost we assume: A1. action rewards $R(s,a)$ are all positive costs denoted $c(s,a) \geq 0$; A2. terminal costs are non negative and goal costs are all zero $c(s_G) = 0$. Doing so permits us to cover a large range of applications as discussed in Section 3.

problem	$V^\star(s_0)$	% rel. states	$h_{\min}(s_0)$	#layers	MDP type
large	33.40	2.27	n.a.	464	M1
larger	68.44	39.35	n.a.	645	M1
c-large	19.99	1.89	19.98	386	M1; A1-A2
c-larger	19.99	50.38	19.98	5262	M1; A1-A2
c-large-bi	19.99	34.51	19.98	2552	M2; A1-A2
c-larger-bi	19.99	50.38	19.98	5189	M2; A1-A2
large-bi	25.99	2.25	n.a.	468	M2
larger-bi	68.43	39.35	n.a.	572	M1

Table 1: Information about problem instances: expected value, percentage of relevant states, heuristic value where $h_{\min}(s_0) = \min_{a \in A(s)} c(s,a) + \min_{s' \in S} h_{\min}(s')$ (Bonet & Geffner 2003a), and number of layers. Abbreviation M1, M2 and A1-A2 stand for layered, weakly coupled and MDPs with positive cost, respectively.

As prioritization solvers attempt to avoid unfruitful and redundant backups, we focus on the speed of convergence to an ε-optimal. We executed all algorithms within the same framework, allowing us to count execution time and the number of executed backups accurately. We distinguish between offline time that corresponds to the time required to build backup orders or compute heuristic values; and the online time that deals with the time needed to find the optimal value function given pre-computed heuristic values or backup orders. All algorithms are coded in JAVA and run on the same Intel Core Duo 1.83GHz processor with 1Gb main memory.

Results

As already mentioned we considered different MDP models that cover a large set of applications, and the results are shown in Table 2,3,4 and 5. The statistic shown that: iTVI-style algorithms outperform the rest of the algorithms in all MDP models. This is mainly because of the good combination of our backup order and the reachability analysis. The backup order used in iTVI achieves the best balance between the number of backups executed and the effort required to prioritized backups. Even though the number of backups may be much more reduced in other prioritization algorithms, in all such methods the benefits do not fully manifest in total execution time.

algorithm	time (sec.) online	time (sec.) offline	time (sec.) total	# backups
large (20424 states, 4 actions)				
Gauß-Seidel VI	119	0	119	10^8
PS (Wingate & Seppi 2005)	97.8	0	97.8	141880
TVI (Dai & Goldsmith 2007)	0.05	6.18	6.23	18864
i-TVI	0.23	0	**0.23**	107800
i-TVI$^+$	0.05	6.18	6.23	**18840**
larger (41856 states, 4 actions)				
Gauß-Seidel VI	371	0	371	10^9
TVI (Dai & Goldsmith 2007)	29.8	104	133.8	**1858100**
i-TVI	31.1	0	**31.1**	6425272
i-TVI$^+$	31.3	104	135	**1858100**

Table 2: Statistics for MDP model M1.

algorithm	time (sec.) online	time (sec.) offline	time (sec.) total	# backups
large-bi (20720 states, 4 actions)				
Gauß-Seidel VI	138	0	138	10^8
PS (Wingate & Seppi 2005)	102	0	102	145212
TVI (Dai & Goldsmith 2007)	0.39	7.96	8.35	99892
i-TVI	0.25	0	**0.25**	110192
i-TVI$^+$	0.05	7.96	8.01	**18864**
larger-bi (41856 states, 4 actions)				
Gauß-Seidel VI	381	0	381	10^{10}
TVI (Dai & Goldsmith 2007)	30.1	105	135.1	**5687312**
i-TVI	32.2	0	**32.2**	5704068
i-TVI$^+$	30.5	105	135.5	**5687312**

Table 3: Statistics for MDP model M2.

algorithm	time (sec.) online	time (sec.) offline	time (sec.) total	# backups
c-large (20424 states, 4 actions)				
Gauß-Seidel VI	22.7	0	22.7	10^8
HDP ($h=0$)	310	0	310	10^9
LRTDP ($h=0$)	2.70	0	2.70	1328413
LAO* ($h=0$)	6.50	0	6.50	998816
HDP ($h=h_{\min}$)	0	21.0	21.0	**20**
LRTDP ($h=h_{\min}$)	2.14	21.0	23.14	1061066
LAO* ($h=h_{\min}$)	4.20	21.0	25.2	632664
TVI (Dai & Goldsmith 2007)	1.43	0.43	1.86	236732
i-TVI	0.60	0	**0.60**	229104
i-TVI$^+$	1.23	0.43	1.66	**191380**
c-larger (41856 states, 4 actions)				
Gauß-Seidel VI	66.4	0	66.4	10^8
HDP ($h=h_{\min}$)	0	66.5	66.5	**20**
TVI (Dai & Goldsmith 2007)	52.9	91.3	144	$1.25 \cdot 10^7$
i-TVI	50.8	0	**50.8**	$1.24 \cdot 10^7$
i-TVI$^+$	17.7	91.3	109	**808608**

Table 4: Statistics for MDP model M1 and A1-A2.

algorithm	time (sec.) online	time (sec.) offline	time (sec.) total	# backups
c-large-bi (20720 states, 4 actions)				
Gauß-Seidel VI	26.8	0	26.8	10^8
HDP ($h=h_{\min}$)	0	26.9	26.9	**20**
LRTDP ($h=h_{\min}$)	370	26.9	397	10^9
TVI (Dai & Goldsmith 2007)	16.0	11.2	27.2	4222144
i-TVI	15.7	0	15.7	4233392
i-TVI$^+$	3.33	11.2	**14.5**	**239076**
c-larger-bi (41856 states, 4 actions)				
Gauß-Seidel VI	65.3	0	65.3	10^8
HDP ($h=h_{\min}$)	0	65.6	65.6	**20**
TVI (Dai & Goldsmith 2007)	55	90.5	145	$1.25 \cdot 10^7$
i-TVI	50.3	0	**50.3**	$1.24 \cdot 10^7$
i-TVI$^+$	3.4	90.5	93.9	**810444**

Table 5: Statistics for MDP model M2 and A1-A2.

Discussion

iTVI-style algorithms outperform TVI, because they make use of additional optimizations. First, in larger domain TVI took 184 seconds while iTVI$^+$ and iTVI took only 135 and 31 seconds respectively. This is mainly because our algo-

rithms are able to use a backup sequence over states. The difference becomes more pronounced for non-layered domains, *e.g.,* large-bi. iTVI-style algorithms also outperform prioritized solver like PS. As discussed in depth above, prioritized sweeping methods suffer from computation overhead due to the management of the priority queue. Unlike prioritized sweeping solvers, iTVI uses a stationary backup order, *i.e.,* computed only once. In many cases, PS executes fewer backups, but the overhead of computing priorities makes it slower than iTVI-style algorithms, *e.g.,* see large domain.

We note that many of the state of the art algorithms were unable to converge or were inappropriate for some domains with specific models. Indeed as argued in the beginning of this paper specific algorithms call for specific MDP models. The performance of forward search methods (LRTDP, LAO*, HDP) are pretty bad in our experiments because we allow the obstacle states to be dead ends. As a result if the sampled trajectory includes a dead-end state, in all such schemes the algorithm will stay for a while in that state. Readers interested in the performance of forward search methods in domains that do not include dead-ends can refer to (Bonet & Geffner 2003a). As a result, for many MDP models we do not include here, performances of all algorithms since they are too slow or inappropriate. The performance of forward search methods in MDPs with positive costs (A1-A2) are very bad for two reasons: first of all domains include multiple dead end states; moreover computing the heuristic estimate $h_{\min}$ incurs significant overhead; hence the total execution time is always more than the one needed by improved topological value iteration. Focussed-RTDP (Smith & Simmons 2006) mitigates the problem of dead-end states by cutting off long trial through the *adaptive maximum depth* procedure. However, in accordance with (Smith & Simmons 2006) choosing the right initial depth D for a given problem is not obvious in FRTDP and dynamically adjusting D may raise inefficient backups.

In contrast, iTVI is able to solve general MDPs. Unfortunately, there are some cases where it is not the appropriate method. One of them assumes a simulation model: given any state s and action a, the algorithm has to access a black box model that samples a successor state s' from the outcome distribution. RTDP-style algorithms assume such a representation of the domain dynamics but not iTVI. This is typically the case in domains handling continuous state and action spaces. However, in domains where the dynamics can be written down in some compact representation, iTVI should be used (at least as a subroutine) in order to achieve a good trade-off between efficiency of the sequence of backups and the overhead of prioritizing backups.

Conclusion

We present iTVI (improved Topological Value Iteration) and iTVI$^+$ algorithms — two new topological MDP solvers. In contrast to state-of-the art TVI, our algorithms suggest in addition an ordering that takes into account the degree of influence of states (measured by path length between them and the start state). This acts as a layering function for the states in the layers of the problem skeleton. iTVI and iTVI$^+$ achieve the best balance between the number of backups executed and the effort required to prioritize backups although the provided backup order is not optimal, *i.e.,* it does not minimize the number of backups. The experiments run on different MDP models including layered MDPs, weakly coupled MDPs, and MDPs with positive costs are encouraging. We believe this set be large enough to show the ability of our approach to generalize to other domains as well.

References

Barto, A. G.; Bradtke, S. J.; and Singh, S. P. 1993. Learning to act using real-time dynamic programming. Technical Report UM-CS-1993-002, University of Alberta.

Bonet, B., and Geffner, H. 2003a. Labeled rtdp: Improving the convergence of real-time dynamic programming.

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

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

Dean, T., and Lin, S.-H. 1995. Decomposition techniques for planning in stochastic domains. In *Proceedings of The 14th International Joint Conference Artificial Intelligence*, 1121–1129.

Givan, R.; Dean, T.; and Greig, M. 2003. Equivalence notions and model minimization in markov decision processes. *Artif. Intell.* 147(1-2):163–223.

Hansen, E. A., and Zilberstein, S. 2001. Lao*: A heuristic search algorithm that finds solutions with loops. *Artif. Intell.* 129(1-2):35–62.

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 1999. Spudd: Stochastic planning using decision diagrams. In *UAI*, 279–28. San Francisco, CA: Morgan Kaufmann.

McMahan, H. B., and Gordon, G. J. 2005. Fast exact planning in markov decision processes. In *ICAPS*, 151–160.

Parr, R. 1998. Flexible decomposition algorithms for weakly coupled markov decision problems. In *Proceedings of the 14th Annual Conference on Uncertainty in Artificial Intelligence*, 422–43.

Peng, J., and Williams, R. J. 1993. Efficient learning and planning within the dyna framework. *Adapt. Behav.* 1(4):437–454.

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

Sutton, R., and Barto, A. 1998. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.

Sutton, R. S. 1991. Planning by incremental dynamic programming. In *ML*, 353–357.

Wingate, D., and Seppi, K. D. 2005. Prioritization methods for accelerating mdp solvers. *J. Mach. Learn. Res.* 6:851–881.