

Topological Orders Based Planning for Solving POMDPs

Jilles S. Dibangoye
Greyc UMR 6072 — DAMAS
Université Laval, Q.C., Canada
jilles-steeve.dibangoye.1@ulaval.ca

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

Abdel-illah Mouaddib
Greyc UMR 6072
Université de Caen, France
mouaddib@info.unicaen.fr

Abstract

Although partially observable Markov decision processes (POMDPs) have received significant attention in past years, to date, solving problems of realistic order of magnitude remains a serious challenge. In this context, techniques that accelerate fundamental algorithms have been a main focus of research. Among them prioritized solvers suggest solutions to the problem of ordering backup operations. Prioritization techniques for ordering the sequence of backup operations considerably reduce the number of backups needed, but involve significant overhead. This paper introduces a novel prioritized method, namely topological order-based planning (TOP), that exploits causal relations between states to deal with two key issues. First, TOP detects the structure of POMDPs as a means of overcoming both the dimensionality and the history curses. Second, it circumvents the problem of unnecessary backups and builds approximate solutions based on a topological order induced by the underlying structure. Empirical experiments prove that TOP is competitive with the best techniques on general domains, and can perform significantly better on layered ones.

Introduction

POMDPs provide a powerful framework for planning in domains involving hidden states and uncertain action effects. However, despite the extensive effort in developing more tractable algorithms (Roy, Gordon, & Thrun 2005; Poupart & Boutilier 2004; Pineau, Gordon, & Thrun 2003; Cassandra, Littman, & Zhang 1997), computational considerations limit the usefulness of POMDPs in real-world-size problems.

The intractability of these techniques is, to a large extent, a consequence of two strongly connected reasons: the well-known *curse of dimensionality* (Bellman 1957), where the dimensionality of the planning problem is driven by the number of states, and the less-well-known *curse of history* (Pineau, Gordon, & Thrun 2003), where the number of histories grows exponentially with the planning horizon. Many attempts have been made to whittle down the set of histories considered for either exact (Cassandra, Littman, & Zhang 1997) or approximate planning strategies (Pineau, Gordon, & Thrun 2003). Exact approaches searching for ϵ -optimal policies over the entire domain, *i.e.*, state and belief

spaces, use pruning techniques to reduce the set of histories. Nevertheless, due to the continuum of the belief space, these approaches turn out to be inefficient in practice. More promising methods for finding approximate solutions in very large domains rely on point-based algorithms. In particular, (Pineau, Gordon, & Thrun 2003) have abandoned full value function backups in favour of only point-based value function backups in *point based value iteration* (PBVI) algorithm. By backing up only discrete belief states, the backup operator is polynomial instead of exponential. However, most point based methods including PBVI still remain constrained by the curse of dimensionality.

Addressing both dimensionality and history curses in the same computational mechanism is the rationale behind our proposed solution method. TOP generalizes the causal relation between states, that has been successfully used in MDPs (Dai & Goldsmith 2007; Bonet & Geffner 2003; Abbad & Boustique 2003). It alleviates the curse of dimensionality by partitioning the state space into mutually causally related states, *i.e.*, layers. The transition graph over these layers provides the problem structure. Then, it completes the planning task over relevant layers and only in the reverse of the topological order induced by the problem structure. It overcomes the curse of history by restricting the search to a small number of layers at each trial. TOP traverses together both the belief space and the underlying MDP space following the topological order. It performs backups only if the value function is likely to change nontrivially if backed up. Doing so circumvents the problem of unfruitful backups. Furthermore, TOP is able to identify layers where the value function has converged and thus re-directs the search towards non-explored layers.

Background and Related Work

We begin with an overview of MDPs, POMDPs and the belief space. We then provide a short introduction to the state-of-the-art approaches.

A Markov Decision Process (MDP) is a tuple (S, A, T, R, γ) where: S is a discrete and finite state space; A is a discrete and finite action space; $T(s'|s, a)$ 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 ; γ is a

discount factor.

The MDP model has then been generalized to the Partially Observable Markov Decision Process (POMDP) for planning in domains involving hidden states and uncertain action effects. More formally a POMDP is a tuple $(S, A, T, R, \Omega, O, \gamma, b_0)$ where: S, A, T, R are the same as in an MDP; Ω is a discrete and finite set of observations; $O(o|s', a)$ is the probability of observing o after executing a and reaching state s ; One key insight in POMDPs is the concept of belief, describing a probability distribution over states and providing a sufficient statistic for a given history. b_0 defines the initial belief state, *i.e.*, the agent belief over its initial state. The next belief state, denoted $b' = \tau(b, a, o)$, that incorporates the latest action-observation pair (a, o) and the current belief state b , is updated as follows:

$$b'(s') = \frac{O(o|s', a) \sum_s b(s) T(s'|s, a)}{pr(o|b, a)} \quad (1)$$

where $pr(o|b, a) = \sum_s b(s) \sum_{s'} T(s'|s, a) O(o|s', a)$. Given a POMDP, the goal is to find a value function V that is a mapping from the belief space to a real-line. Moreover, since V is proven piecewise linear and convex (Sondik 1978), it can be represented as a set of α vectors, *i.e.*, $V = \{\alpha_1, \alpha_2, \dots, \alpha_m\}$, such that: $V(b) = \max_i \alpha_i \cdot b$. In addition, V can be iteratively computed using point based backup operations (Pineau, Gordon, & Thrun 2003; Spaan & Vlassis 2005; Smith & Simmons 2004; 2005):

$$g_{a,o}^\alpha(s) = \sum_{s'} O(o|s', a) T(s'|s, a) \alpha_i(s') \quad (2)$$

$$g_a^b = r_a + \gamma \sum_o \arg \max_{g_{a,o}^\alpha: \alpha \in V} (b \cdot g_{a,o}^\alpha) \quad (3)$$

$$backup(b) = \arg \max_{g_a^b: a \in A} (b \cdot g_a^b) \quad (4)$$

where $r_a(s) = R(s, a)$ is a vector representation of the reward function.

Approximate Solvers

The computational hurdle posed by the number of vectors that are generated for computing an optimal value function over the entire belief simplex has led to the development of a wide variety of approximate methods.

Instead of computing an optimal value function over the entire belief space, one possible approximation does so only over a finite subset B of belief states (Lovejoy 1991). As a result, the backup operator is polynomial and the complexity of the value function is bounded by the number $|B|$ of belief states. Nevertheless, an optimal value function over B may not generalize well for the belief states not in B , depending on the belief selection method.

In contrast, point-based algorithms backup only discrete belief states that are reachable from the starting belief state b_0 (Pineau, Gordon, & Thrun 2003; Spaan & Vlassis 2005; Smith & Simmons 2004; 2005). By backing up only reachable belief states, it is empirically and theoretically proven that the computed value function will generalize well in other belief states. Techniques of this family differ only through the method used to generate the reachable belief states.

In particular, point-based value iteration (PBVI) computes alternatively both a belief set B and the optimal value function over B (Pineau, Gordon, & Thrun 2003). The algorithm starts by selecting the initial belief set B_0 including the initial belief b_0 , computes the optimal value function over B_0 , and then expands B_0 to B_1 with all the most distant immediate successors of B_0 and computes again the optimal value function over B_1 . This process is repeated until some stopping criteria is reached. Many drawbacks, however, limit PBVI to scale up to larger domains. First, the expansion step of the belief set B incurs considerable computation efforts. Indeed, this step consists in comparing $|B|$ belief states to their $|B||A||O|$ immediate successors with respect to a distance metric, *e.g.*, L_2 . Hence, in the worst case the sub-routine requires $O(|B|^2|A||O||S|)$ operations. To cope with this weak point (Pineau, Gordon, & Thrun 2003) also suggested sampling a successor for each belief-action pair, reducing the complexity of the expansion step to $O(|B|^2|A||S|)$. The second point of weakness is the update phase of PBVI. This is mainly because PBVI proceeds by backing up all beliefs B at each loop even if unnecessary or redundant.

PERSEUS (Spaan & Vlassis 2005) is another point-based algorithm that is close in spirit to PBVI. Except that the expansion step is replaced by a random walk starting at the initial belief state b_0 . In addition, instead of backing up all belief states at each loop, PERSEUS backups only a subset of the selected belief states. This is the subset of beliefs whose values have not been improved by the previous vector update, during the current loop. Since PERSEUS often updates only a subset of belief states, it may either converge much faster to an approximate policy or use a larger set of beliefs to improve its value function quality. The major problem with PERSEUS consists in its slow rate of convergence as illustrated in the experiments. Although the random walk directs the search towards more likely reachable parts of the belief simplex, there is no guarantee that this will provide paths towards rewards. Moreover, by traversing the belief space through the random walk its behavior is random and unpredictable, and so are its performances on some tested domains.

Of this family of algorithms, *heuristic search value iteration* (HSVI) has impressive performance on a number of larger domains (Smith & Simmons 2004; 2005). HSVI proceeds by creating sequences of belief states, *i.e.*, traversals, based on both upper and lower bounds over the value function, denoted \bar{V} and \underline{V} respectively. Starting with the initial belief state b_0 , the next belief state is the successor belief state of the previous one which maximizes the gap between the bounds. The belief expansion stops when some criteria is reached, afterwards HSVI backups the sequence backwards. Unfortunately, both maintaining the upper-bound and selecting the next belief state incur considerable overhead. Hence, the gain resulting from the reduced number of backups does not fully manifest in the total execution time. As the algorithm proceeds, the gap between bounds over the initial belief b_0 gets smaller and smaller, HSVI is stopped when the gap is less than a small threshold ϵ . This provides a theoretical bound on the quality of the computed value function

over the visited belief states. Nonetheless, from a practical point of view closing this gap is impracticable unless the initial upper bound is close to the optimal value function. For example, in Rock Sample domains the initial value of the upper-bound is close to the approximate value function returned. This is mainly because the transition probabilities in Rock Sample domains are deterministic.

Trial-based Solvers

To allow some belief states to be updated more than others, researchers turned attention to trial-based methods. These are techniques where simulation trials are executed, creating traversals of belief states. Among them *forward search value iteration* (FSVI) has demonstrated good performances (Shani, Brafman, & Shimony 2007). The FSVI algorithm uses the underlying MDP Q -value function as a heuristic to traverse the belief simplex. It maintains both the underlying MDP state s and the belief state b , allowing the action a selection based on the current state and the MDP Q -value function. FSVI proceeds then by selecting the next state s' and observation o by sampling them from $T(\cdot|s, a)$ and $O(\cdot|a, s')$ respectively. The advantage with FSVI is that the overhead of computing the MDP Q -value function is negligible and so is the action selections. One major drawback is its inability to recognize traversals that will improve its state information. In fact, by making use of the optimal Q -value function of the underlying MDP, FSVI limits its ability to create traversals that will visit states that may provide useful observations. Even more importantly, it fails to re-direct the search towards states that are probably unreachable following the underlying MDP Q -value function, even if they may provide additional rewards. As a consequence, FSVI results in a sub-optimal value function (over visited beliefs) in certain domains. Nevertheless, in domains that do not require long-term information gathering tasks and/or a broader exploration of the belief simplex, FSVI is very competitive. Yet even a very conservative and simple action exploration policy, e.g., ϵ -greedy, can balance the lack of exploration.

Prioritized Solvers

Prioritization techniques address the problem of finding quickly and accurately an approximate value function by means of ordering backup operations and circumventing the problem of unnecessary backups. The idea of arranging backups in a good order has already been discussed before by (Shani, Brafman, & Shimony 2006; Virin *et al.* 2007) and to some extent by (Shani, Brafman, & Shimony 2007; Smith & Simmons 2004; 2005).

Shani *et al.* (Shani, Brafman, & Shimony 2006) suggested the *prioritized value iteration* (PVI) algorithm. PVI uses a fixed set B of belief states and backed up these belief states in the order of the Bellman error, defined as follows: $e(b) = \max_a [r_a \cdot b + \sum_o pr(o|b, a)V(\tau(b, a, o))] - V(b)$. The PVI algorithm backups at each step the belief $b \in B$ that is more likely to improve considerably its value function. To do so, Shani *et al.* suggest, in the clean version of PVI choosing the belief that yields the highest Bellman error. The algorithm stops when it fails to find a belief with non-zero error. By backing up beliefs in the order of the Bell-

man error, it is likely that some belief states will be backed up more often than others. This may result in a significant computational gain. Unfortunately, computing the Bellman error after each backup incurs considerable computational efforts. The complexity of the update of the Bellman error is non-negligible with respect to the complexity of the backup operation: in the worst-case they require $O(|B||A||O||S|)$ and $O(|S|^2|V||A||O|)$ respectively. To overcome this issue, Shani *et al.* computed the Bellman error only on a sample subset \tilde{B} of belief states, reducing the complexity of the update of the Bellman error to $O(|\tilde{B}||A||O||S|)$.

In the same vein, Virin *et al.* (Virin *et al.* 2007) implemented a prioritized algorithm, namely *soft cluster based value iteration* (SCVI), inspired by generalized prioritized sweeping methods proposed in MDP literature (Wingate & Seppi 2005). To leverage the complexity of computing good backup sequences, similarly to the FSVI algorithm they use a clustering algorithm based on the underlying MDP optimal solution. Then, SCVI backups belief states in order of decreasing relevance to the current cluster. This method has the advantage of avoiding the systematic queue sorting procedures required in PVI. However, once again since it is very likely that a belief state will have some belief over states in a number of clusters, each belief state can be updated at each iteration, even if it is not required.

Finally, one can look at HSVI and FSVI as prioritization methods. Indeed, both proceed by traversing the belief simplex following heuristics and perform backups backwards. Executing backups in the reverse order of the traversal has important consequences. As we later discuss, beliefs and their value functions are causally related. As such, the value function of a successor belief state may contribute to the value function of the current belief state. Thus, backing up successor belief states first can result in a speed up of the value function convergence.

Topological Order based Planning

In this section, we provide a high-level description of the proposed approach. We represent a POMDP M as a finite set of smaller POMDPs $\{M_k\}_{k=0}^K$, such that each POMDP M_k is defined by a tuple $\langle S_k, A, O, T, \Omega, B_k, \gamma \rangle$, where: for all $k = 0, 1, \dots, K$, the set S_k is a state subspace of POMDP M state space S , we also refer to S_k as a *layer*. We denote B_k the set of beliefs visited when traversing layer S_k . Since a belief $b \in B_k$ can be partially out of the current layer S_k , we use the state $s \in S_k$ of the environment the agent is at within simulation to determine whether or not the belief b belongs to B_k . Hence, our traversals consist off state-belief pairs (s, b) . Moreover, we introduce two backup orders \prec_1 and \prec_2 over belief subspaces $\{B_k\}_{k=0}^K$ and belief states in a single subspace B_k respectively. We argue that by identifying these backup orders we may avoid redundant backups, and thus quickly and accurately compute good policies. But even more importantly, we show that it is possible to discover such backup orders with a negligible overhead. As a result, the computation gain of prioritizing backups will fully manifest in the total execution time as demonstrated in the experiments.

High-level description

The TOP algorithm is a trial-based solver, see Algorithm 1. Similarly to FSVI or HSVI, it traverses the belief simplex following heuristics and updates beliefs backwards. What make TOP apart from others trial-based solvers are heuristics it uses to build traversals of state-belief pairs (s, b) . In fact, TOP begins by identifying the POMDP layers $\{S_k\}_{k=0}^K$ and their dependencies. Then, using the underlying MDP state s (from pair (s, b)), it builds paths as short as possible from an initial layer to a *solvable* one. A layer S_k is said to be solvable if it is either a layer with no successor layers or a layer that is causally related to already solved layers. Otherwise, it is said to be non-solvable. A solvable layer can also be seen as a *goal* layer. As TOP proceeds, it maintains an optimistic estimate of the contribution of each solvable layer in the value function using the Bellman error. As a result, if this estimate for a given solvable layer S_k is less or equal to ε then the layer is said to be solved and $\forall s \in S_k, s.SOLVED = true$ (initially $\forall s \in S_k, s.SOLVED = false$). It turns out that the TOP algorithm solves each layer of the POMDP only after successor layers have been solved. Finally, the algorithm stops when all layers have been solved or when the pre-defined average discounted reward (ADR) is reached.

Algorithm 1: TOP for solving POMDPs.

```

TOP( $s_0, b_0$ )
begin
  repeat
    TOPTRIAL( $s_0, b_0$ )
  until  $V$  has not converged
end

TOPTRIAL( $s, b$ )
begin
  if  $\neg s.SOLVED$  then
     $s^* \leftarrow \text{PICKNEXTSTATE}(s, a)$ 
     $a^* \leftarrow \text{PICKACTION}(s, s^*)$ 
     $o^* \leftarrow \text{PICKNEXTOBSERVATION}(s^*, a)$ 
    if  $s^*.TERMINAL$  then break
    TOPTRIAL( $s^*, \tau(b, a^*, o^*)$ )
    UPDATE( $s, b, V$ )
  end
end

```

Prioritizing Belief Subspaces $\{B_k\}_{k=0}^K$

The problem of determining a backup order \prec_1 over subspaces of the belief simplex can be viewed as the problem of identifying a causal dependency between these belief subspaces. Following the above intuition, we first observe that belief states and their value functions are causally related. To explain this insight, let us consider POMDP M , belief subspaces B_{k+1} and B_k . If any belief $b' \in B_{k+1}$ is a successor belief of some belief $b \in B_k$ for an action-observation pair (a, o) , then $V(b)$ is dependent on $V(b')$. Indeed, the value function of b can be iteratively updated as follows: $V(b) = \max_a (b \cdot r_a + \gamma \sum_o pr(o|a, b) V(\tau(a, o, b)))$. For this reason, we want to backup beliefs $b' \in B_{k+1}$ ahead of beliefs $b \in B_k$, i.e., $B_{k+1} \prec_1 B_k$. This causal relation is transitive. However, POMDPs are cyclic, and causal relations are very common among belief subspaces. In this case, we turn our attention to the problem of finding a backup order \prec_1 over belief subspaces of a cyclic POMDP. One basic

idea, successfully applied in MDPs and extended here, is the following (Dai & Goldsmith 2007; Bonet & Geffner 2003; Abbad & Boustique 2003): Group states that are mutually causally related together and construct *layers* $\{S_k\}_{k=0}^K$; Let these layers form a new POMDP M' . Then, M' is no longer cyclic. In this case, the backup order \prec_1 is given by the reverse topological order of states in M' . More precisely, we can backup the belief subspaces $\{B_k\}_{k=0}^K$ in only one *virtual* iteration in the order of \prec_1 . However, we are still left with the problem of finding layers $\{S_k\}_{k=0}^K$ that induced belief subspaces $\{B_k\}_{k=0}^K$. Recent research has shown that the problem of finding layers is reduced to the problem of finding the strongly connected components in the MDP graph, thus in the POMDP graph G where observation edges are omitted (Dai & Goldsmith 2007; Bonet & Geffner 2003; Abbad & Boustique 2003). There exist many algorithms, including the Tarjan algorithm (Tarjan 1972), that find layers in only a linear time.

Prioritizing Backups in B_k

This section aims at prioritizing the order beliefs in a single layer should be updated. This is particularly useful for domains with a single layer, e.g., HALLWAY. In fact, the backup order \prec_1 discussed previously is unable to determine the order in which beliefs in a single subspace B_k should be backed up. This is mainly because all states in a single layer S_k are causally mutually related, thus all beliefs in B_k are more likely to be mutually causally related too. As argued by Shani *et al.* (Shani, Brafman, & Shimony 2006), the update of a single belief state may affect the value function of the entire belief space, and usually improves its mutually causally related belief states. To handle this issue we rely on a forward search scheme: (1) creating together trajectories of both beliefs and the underlying MDP states, similarly to what the FSVI algorithm does; (2) prioritizing the order in which beliefs in B_k are generated and updated. The latter point provides the backup order \prec_2 used to prioritize beliefs in a single subspace B_k .

The heuristic the TOP algorithm used to traverse the belief space is based on the POMDP structure and the true state of the environment the agent is at within simulation. This heuristic seeks to generate traversals of state-belief pairs towards rewards, then updates beliefs backwards. To achieve this, one can use the underlying MDP Q -value function to direct the search towards states with high estimate values, as suggested by FSVI. Unfortunately, this strategy fails to update the estimate value of a state during the course of the search. Hence, it systematically tracks the same traversals.

State priorities Here, we suggest a dynamically adjusted heuristic that is based on state priorities. To build the order in which belief $b \in B_k$ should be backed up, we store additional information in the model. Each state remembers its *predecessors*, i.e., the states that have a non-zero transition probability to it under some action. In addition, each state s has a *priority* $pri(s)$, initially set to the underlying MDP value function. This will be used as a heuristic that directs the search towards regions of the state space with high priority. Furthermore, we perform a backup for the

belief b , associated with a state-belief pair (s, b) , only if state s yields a priority greater than ϵ . Indeed, the state priority gives an estimate of the gain of backing up the associated belief state. Note that a state can be associated to many different belief states. Starting with an initial state-belief pair (s_0, b_0) , the update sub-routine works as described in Algorithm 2. Let us denote $\hat{p}(s, s')$ the

Algorithm 2: UPDATE sub-routine.

```

UPDATE( $s, b, V$ )
begin
  if  $pri(s) > \epsilon$  then
    update the belief state  $b: V' \leftarrow add(V, backup(b))$ 
    set state  $s$  priority  $pri(s)$  back to 0.
    compute the error  $\Delta$  of the change  $V'(b) - V(b)$ 
    use  $\Delta$  to modify the priorities of predecessors of  $s$ 
end

```

highest transition probability of reaching s' starting in s , $\hat{p}(s, s') = \max_{(a', o') \in A \times \Omega} T(s'|s, a') \cdot O(o'|s', a')$. If we have updated value function V for state-belief pair (s', b) and it has changed by the amount of Δ , the immediate predecessors of s' are informed of this event. Any state s for which there exists an action-observation pair (a, o) such that $\hat{p}(s, s') \neq 0$ has its priority $pri(s)$ promoted to $\Delta \cdot \hat{p}(s, s')$ unless its priority already exceed that value.

Generating prioritized traversals At each trial of the TOP algorithm, our goal is to find a short traversal that yields a high cumulative priorities. This is because such traversals direct the search towards relevant regions of the belief simplex. Given a state-belief pair (s, b) where $s \in S_k$, the following describes heuristics we use to find the successor state-belief pair (s^*, b^*) . Roughly, we proceed by determining a state-action-observation tuple (s^*, a^*, o^*) then induce the next belief b^* as follows $b^* = \tau(b, a^*, o^*)$. We use slightly different heuristics for each type of layer:

If the current state-belief pair (s, b) belongs to a non-solvable layer, we seek to generate a path as short as possible from s to a state s_G of a solvable layer. The state s_G is in fact a state with non-zero priority and serves as a temporary sub-goal in its associated solvable layer. Given the graphical representation G of a POMDP, finding the shortest path from any pair of states is equivalent to the problem of determining the shortest paths for all pairs of vertices $(s, s') \in S^2$, where the graph is G and each arc is labeled with a cost defined as follows:

$$cost(s, s') = \begin{cases} 1 - \hat{p}(s, s') & \text{If } \hat{p}(s, s') > \epsilon \\ +\infty & \text{Otherwise.} \end{cases}$$

For solving such problem one can use the Floyd-Warshall algorithm (Cormen *et al.* 2001) that runs in time $O(|S|^3)$ and returns both the shortest distance between two states $dist(s, s')$ and the next state $next(s, s')$ in the shortest walk from s to s' . As a result, given the current state s and sub-goal s_G , the next state is given by $next(s, s_G)$. The overhead of running Floyd-Warshall algorithm is negligible given the complexity of POMDPs.

On the other hand, if the current state-belief pair (s, b) belongs to a solvable layer S_k , the traversals are generated in a prioritized fashion: we first find state s_G with

the highest priority in S_k ; then, we use the results of the Floyd-Warshall algorithm and return state $next(s, s_G)$ as described in Algorithm 3. In any case, we use the next state s^* resulting from PICKNEXTSTATE sub-routine to determine which action-observation pair (a, o) to simulate next. Given the next state s^* , we use two possible strategies: (1) the simple one is to sample the action-observation pair (a, o) from $T(s^*|s, \cdot)$ and $O(\cdot|s^*, a)$ respectively; (2) the second one consists in selecting a pair (a^*, o^*) that maximizes the outcome of reaching state s^* starting from s : $(a^*, o^*) = \arg \max_{a, o} \tau(b, a, o)[s^*] \cdot pri(s^*)$. The trial is ended when the next state s^* belongs to a layer that has already been solved, *i.e.*, $s^*.TERMINAL = true$. A layer is said to be solved if all the related states have their priority set at zero. This is motivated by the fact that state priorities describe the changes in the value function, hence a layer is solved if the value function has converged among beliefs in that layer.

Algorithm 3: PICKNEXTSTATE sub-routine.

```

PICKNEXTSTATE( $s, b$ )
begin
  pick any state  $s_G$  that belongs to a solvable layer
  if  $\neg s.SOLVABLE$  then for all  $s' \in S: s'.SOLVABLE$  do
    if  $dist(s, s_G) > dist(s, s')$  then  $s_G \leftarrow s'$ 
    else if  $dist(s, s_G) = dist(s, s')$  then
      if  $pri(s_G) < pri(s')$  then  $s_G \leftarrow s'$ 
  else find state  $s_G$  with the highest priority
   $s^* \leftarrow next(s, s_G)$ 
end

```

Empirical Evaluations

We now evaluate the performance of TOP in comparison with other recent forward search algorithms such as HSVI (Smith & Simmons 2005), and FSVI. In addition, we have also added some solvers including PVI, a prioritized POMDP solver (Shani, Brafman, & Shimony 2006) and PBVI (Pineau, Gordon, & Thrun 2003). We use Guy Shani's source code that implements all these algorithms in JAVA and the experiments have been run on the same Intel Core Duo 1.83GHz CPU processor with 1Gb main memory.

We evaluate the following criteria: *Value function evaluation* – Average discounted reward (ADR): $\frac{\sum_{i=0}^{\#trials} \sum_{j=0}^{\#steps} \gamma^j r_j}{\#trials}$. ADR is filtered using the first order filter to reduce the noise in ADR. As we are interested in the speed of convergence, each solver was executed until its value function produced a pre-defined ADR. We also report the value of the initial belief b_0 demonstrating different effects of each strategies on the value function structure; *Execution time* – we report the CPU time but as this is an implementation specific measurement, we also report the amount of backup operations; *Memory* – we also show the number of vectors in the final value function.

We tested all algorithms over several well known benchmarks from the POMDP community, including: TigerGrid (Cassandra, Kaelbling, & Littman 1994); navigation problems, *e.g.*, Hallway, Hallway2 and Mit (Cassandra, Kaelbling, & Littman 1994); Rock Sample domains (Smith & Simmons 2004).

Results

Method	ADR	$V(b_0)$	$ V $	Time (sec)	#Backups
TigerGrid					
HSVI	0.68	1.75	1337	535	1695
FSVI	0.66	1.89	1608	380	1275
PBVI	0.66	2.15	472	3749	33101
PVI	0.60	0.69	357	> 3530	480
TOP	0.68	1.05	755	115	640
Hallway					
HSVI	0.51	0.88	258	391	886
FSVI	0.51	0.89	317	440	969
PBVI	0.51	0.75	63	59	1447
PVI	0.51	0.69	185	3272	500
TOP	0.52	0.60	213	56	352
Hallway 2					
HSVI	0.35	0.16	128	126	145
FSVI	0.36	0.24	252	76	204
PBVI	0.35	0.26	128	347	1355
PVI	0.35	0.27	307	> 9565	400
TOP	0.35	0.12	100	21	98
Rock Sample (4, 4)					
HSVI	18.036	17.92	173	11	239
FSVI	18.036	13.63	79	7	98
TOP	18.036	13.63	40	3	68
Rock Sample (5, 5)					
HSVI	19.237	19.24	50	22	60
FSVI	17.840	13.84	185	200	307
TOP	19.204	17.20	61	37	88
Rock Sample (5, 7)					
HSVI	24.202	22.96	208	2754	461
FSVI	22.198	14.97	353	4057	384
TOP	24.086	21.19	317	2791	436
Mit					
HSVI	0.88	0.77	1786	17344	2288
FSVI	0.86	0.69	339	1213	1275
TOP	0.88	0.44	520	820	385

Table 1: Performance measurements.

Like the TOP algorithm, both HSVI and FSVI use belief traversals and differ only in the way they select and order the sequence of backups over these beliefs. As Table 1 shows, TOP is faster than HSVI, FSVI over all domains. At the moment our experimental results are only preliminary and should not be used for comparison yet. In fact, it seems that our implementation of FSVI is not well optimized. We will soon report more detailed experimental results. Nevertheless, in all benchmarks, TOP executes the few backups that require almost no effort in comparison to the other solvers. Even more importantly, by performing backups only when the priority of the underlying MDP state is greater than a small number ϵ , TOP drastically reduces the number of useless backups. We notice, however, that although the ADRs of all solvers was roughly tied for all domains, the value of the initial belief state was quite different from one solver to another; see for example RockSample (5, 7) Table 1. This can be explained by the setting in these benchmarks where the model definition does not define a terminal state. We also noted that the FSVI algorithm suffers from a lack of exploration. Indeed, the FSVI algorithm, as already mentioned, uses the underlying MDP Q -value function to traverse the belief space. Unfortunately, since the Q -value function is fixed FSVI keeps track on the same traversals, hence it is unable to improve the value of the initial belief. Yet even a very conservative and simple action exploration policy balances that for the benchmarks tested. The TOP algorithm is also faster than POMDP solvers that use a fixed set of beliefs including PBVI, PVI and PERSEUS algorithms. We did not report the performances of PERSEUS because it exceeds time limits in all tested domains. As we are interested in

the speed of convergence, each algorithm was executed until its value function produced a predefined ADR, no matter the number of beliefs it required. While PBVI proceeds by backing up a set of belief states then expands this set and so on, PVI tries to prioritize its sequence of backups resulting in a non-negligible computation overhead. Looking at their performances in comparison to TOP, we observe that both require more time than TOP to meet the predefined ADR. Although PVI may perform a small number of backups, *e.g.*, TigerGrid Table 1, the effort required to prioritize the sequence of belief states is prohibitive.

Conclusion

We have introduced a new POMDP solution method, namely topological order-based planning (TOP), that studies the structural properties of POMDPs to find the best back up orders. This heuristic is based on the idea that different POMDPs have different graphical structures, and graphical structure of a POMDP intrinsically determines the complexity of solving that POMDP. Although extensive efforts have been made to solve real-world-size POMDPs, we note that none of the state-of-the-art POMDP solution methods makes use of this principle to guide belief backups. As an alternative, TOP offers many advantages. First, it finds a solution of desired quality by backing up only when necessary. Second, it backups the belief subspaces defined only over state subspaces. This enables it to tackle both dimensionality and history curses in the same computational mechanism.

References

- Abbad, M., and Boustique, H. 2003. A decomposition algorithm for limiting average markov decision problems. *Oper. Res. Lett.* 31(3):473–476.
- Bellman, R. E. 1957. *Dynamic Programming*. Dover Publications, Incorporated.
- Bonet, B., and Geffner, H. 2003. Faster heuristic search algorithms for planning with uncertainty and full feedback. In *IJCAI*, 1233–1238.
- Cassandra, A. R.; Kaelbling, L. P.; and Littman, M. L. 1994. Acting optimally in partially observable stochastic domains. In *AAAI*, 1023–1028.
- Cassandra, A.; Littman, M. L.; and Zhang, N. L. 1997. Incremental Pruning: A simple, fast, exact method for partially observable Markov decision processes. In *UAI*, 54–61.
- Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; and Stein, C. 2001. *Introduction to Algorithms, Second Edition*. The MIT Press.
- Dai, P., and Goldsmith, J. 2007. Topological value iteration algorithm for Markov decision processes. In *IJCAI*, 1860–1865.
- Lovejoy, W. S. 1991. Computationally feasible bounds for partially observable markov decision processes. *Operations Research* 39:175–192.
- Pineau, J.; Gordon, G.; and Thrun, S. 2003. Point-based value iteration: An anytime algorithm for POMDPs. In *NIPS*.
- Poupart, P., and Boutilier, C. 2004. VDCBPI: an approximate scalable algorithm for large POMDPs. In *NIPS*.
- Roy, N.; Gordon, G.; and Thrun, S. 2005. Finding approximate pomdp solutions through belief compression. *JAIR* 23:1–40.
- Shani, G.; Brafman, R. I.; and Shimony, S. E. 2006. Prioritizing point-based pomdp solvers. In *ECML*, 389–400.
- Shani, G.; Brafman, R. I.; and Shimony, S. E. 2007. Forward search value iteration for pomdps. In *IJCAI*, 2619–2624.
- Smith, T., and Simmons, R. 2004. Heuristic search value iteration for pomdps. In *UAI*, 520–527.
- Smith, T., and Simmons, R. G. 2005. Point-based POMDP algorithms: Improved analysis and implementation. In *UAI*.
- Sondik, E. J. 1978. The optimal control of partially observable markov decision processes over the infinite horizon: Discounted cost. *Operations Research* 12:282–304.
- Spaan, M. T. J., and Vlassis, N. 2005. Perseus: Randomized point-based value iteration for POMDPs. *JAIR* 24:195–220.
- Tarjan, R. 1972. Depth-first search and linear graph algorithms. *SIAM J. Comput.* 1(2).
- Virin, Y.; Shani, G.; Shimony, S. E.; and Brafman, R. I. 2007. Scaling up: Solving pomdps through value based clustering. In *AAAI*.
- Wingate, D., and Seppi, K. D. 2005. Prioritization methods for accelerating mdp solvers. *J. Mach. Learn. Res.* 6:851–881.