APPEARED IN
Int. Conf. on Automated Planning & Scheduling, Workshop on Solving Real-World POMDP Problems, 2010

A POMDP Approach to Robot Motion Planning under Uncertainty

Yanzhu Du[1*] David Hsu[2] Hanna Kurniawati[3] Wee Sun Lee[2] Sylvie C.W. Ong[2] Shao Wei Png[4]

[1]Department of Computer Science
Stanford University
Stanford, CA 94305, USA

[2]Department of Computer Science
National University of Singapore
Singapore, 117417, Singapore

[3]Singapore-MIT Alliance for Research & Technology
Singapore, 117543, Singapore

[4]School of Computer Science
McGill University
Montreal, Quebec H3A 2A7, Canada

Abstract

Motion planning in uncertain and dynamic environments is critical for reliable operation of autonomous robots. Partially observable Markov decision processes (POMDPs) provide a principled general framework for such planning tasks and have been successfully applied to several moderately complex robotic tasks, including navigation, manipulation, and target tracking. The challenge now is to scale up POMDP planning algorithms and handle more complex, realistic tasks. This paper outlines ideas aimed at overcoming two major obstacles to the efficiency of POMDP planning: the "curse of dimensionality" and the "curse of history". Our main objective is to show that using these ideas—along with others—POMDP algorithms can be used successfully for motion planning under uncertainty for robotic tasks with a large number of states or a long time horizon. We implemented some of our algorithms as a software package *Approximate POMDP Planning Library* (APPL), now available for download at `http://motion.comp.nus.edu.sg/projects/pomdp/pomdp.html`.

Introduction

Motion planning research has made great strides in the last decades. Using probabilistic sampling, motion planning algorithms can now effectively deal with robots with many degrees of freedom (DoFs) (Choset et al. 2005). However, these algorithms often assume that the robot can *perfectly* control its actions and sense the ambient environment. This assumption is reasonable in carefully engineered settings, such as robot manipulators on manufacturing assembly lines, but becomes more and more difficult to justify, as robots venture into new application domains in homes or in offices. In these uncontrolled, uncertain, and dynamic environments, motion planning with imperfect state information is critical for autonomous robots to operate reliably.

Partially observable Markov decision processes (POMDPs) (Smallwood and Sondik 1973) are a principled general framework for such planning tasks. With imperfect state information, the robot cannot decide the best action based on a single current state, as it is unknown. Instead, the robot must consider all possible states consistent with the observations. In a POMDP, we represent such a set of states as a *belief* b, which is a probability distribution over the robot's state space. The probability $b(s)$ of a state s indicates how likely the robot is in s. To solve a POMDP, we reason in the belief space $\mathcal{B}$, the space containing all beliefs, and compute a *policy* that prescribes the best action for every belief that the robot may encounter. POMDP planning systematically takes into account the uncertainty in robot control, sensing, and environment changes, in order to achieve robust performance. It has shown promising performance in several moderately complex robotic tasks (Hsiao, Kaelbling, and Lozano-Pérez 2007; Hsu, Lee, and Rong 2008; Pineau, Gordon, and Thrun 2005; Roy, Gordon, and Thrun 2005).

One common criticism of POMDP planning is its high computational complexity. Solving POMDPs exactly is computationally intractable (Papadimitriou and Tsisiklis 1987). To develop efficient POMDP planning algorithms, we face at least two main obstacles. The first is the "curse of dimensionality". The belief space, which allows us to reason systematically about uncertainty, is also the source of this difficulty. Suppose that a robotic task is modeled with a discrete state space. The corresponding belief space then has dimensionality equal to the number of states for the robot: a robot with 1,000 states results in a belief space of 1,000 dimensions! In recent years, point-based POMDP algorithms have made significant progress in overcoming this difficulty by sampling the belief space probabilistically and computing approximate solutions. The fastest algorithms today, such as HSVI2 (Smith and Simmons 2005) and SARSOP (Kurniawati, Hsu, and Lee 2008), can solve POMDPS with hundreds of states in seconds and POMDPs with up to 100,000 states in reasonable time. However, more progress is needed to handle truly complex robotic tasks.

The second obstacle is the "curse of history". Motion planning tasks often require the robot to take many actions to achieve a goal, resulting in a long time horizon for planning. The complexity of planning usually grows exponentially with the time horizon.

This paper outlines two ideas aimed at overcoming each of the two obstacles. Our main objective is to show that using these ideas—along with others—POMDP algorithms

[*]The work was done when Y. Du, H. Kurniawati, and S.W. Png were with the National University of Singapore.

ICAPS 2010 POMDP Practitioners Workshop, May 12, 2010, Toronto, Canada.

can be used successfully for motion planning under uncertainty for robotic tasks with a large number of states or a long time horizon. We demonstrate this through experiments in simulation on three distinct tasks: grasping, target tracking, and navigation. All these tasks are based on ones reported in the literature in the last two years. We modeled these tasks as POMDPs and solved the resulting POMDPs using our algorithms. Furthermore, we implemented some of our algorithms as a software package, now freely available for download.

Background

Related work

In classic motion planning (without uncertainty), *configuration space* (Lozano-Pérez 1983) provides a general conceptual framework which allows many apparently different motion planning problems to be formulated uniformly. It has also led to the development of efficient planning algorithms based on the idea of probabilistically sampling a robot's configuration space (see (Choset et al. 2005) for a survey). Sampling-based algorithms provide the most successful approach today for motion planning of robots with many DoFs. Although lots of interesting and significant work has been done for motion planning with uncertainty (see (Latombe 1991; LaValle 2006) for surveys), a general framework like configuration space is not yet available. POMDPs (Smallwood and Sondik 1973; Kaelbling, Littman, and Cassandra 1998) can potentially play this role. However, POMDP planning had been plagued with computational efficiency issues. Just ten years ago, the best algorithms could spend hours solving POMDPs with only a dozen states, which are woefully inadequate for modeling realistic robotic tasks. Only recently, point-based POMDP algorithms (Pineau, Gordon, and Thrun 2003; Spaan and Vlassis 2005; Smith and Simmons 2005; Kurniawati, Hsu, and Lee 2008; Ong et al. 2009; Kurniawati et al. 2009) made dramatic progress in computational efficiency so that POMDPs with 10,000s and sometimes 100,000s of states can be solved approximately in reasonable time. These algorithms made it more practical to apply POMDP as a tool for robot motion planning under uncertainty. The ideas and examples in this paper illustrate the kind of robotic tasks that can be handled by the fastest POMDP algorithms today.

POMDPs

A POMDP models an agent taking a sequence of actions under uncertainty to maximize its total reward. Formally a discrete POMDP with an infinite horizon is specified as a tuple $(S, A, O, T, Z, R, \gamma)$, where S is a set of states, A is a set of actions, and O is a set of observations.

In each time step, the agent takes an action $a \in A$ and moves from a state $s \in S$ to $s' \in S$. Due to the uncertainty in action, the end state s' is described as a conditional probability function $T(s, a, s') = p(s'|s, a)$, which gives the probability that the agent lies in s', after taking action a in state s. The agent then makes an observation on the end state s'. Due to the uncertainty in observation, the observation result $o \in O$ is again described as a conditional probability function $Z(s', a, o) = p(o|s', a)$ for $s' \in S$ and $a \in A$.

To elicit desirable agent behavior, we define a suitable reward function $R(s, a)$. In each step, the agent receives a real-valued reward $R(s, a)$, if it is in state s and takes action a. The goal of the agent is to maximize its expected total reward by choosing a suitable sequence of actions. When the sequence of actions has infinite length, we typically specify a discount factor $\gamma \in (0, 1)$ so that the total reward is finite and the problem is well defined. In this case, the expected total reward is $\mathrm{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$, where s_t and a_t denote the agent's state and action at time t.

POMDP planning means computing an *optimal policy* that maximizes the agent's expected total reward. In the more familiar case where the agent's state is fully observable, a policy prescribes an action, given the agent's current state. However, a POMDP agent's state is partially observable and not known exactly. So we rely on the concept of *belief state*, or *belief* for short. A belief is a probability distribution over S. A POMDP policy $\pi\colon \mathcal{B} \to A$ maps a belief $b \in \mathcal{B}$ to the prescribed action $a \in A$.

A policy π induces a value function V^π, which specifies the expected total reward $V^\pi(b)$ of executing π starting from b. It is known that V^*, the value function for an optimal policy π^*, can be approximated arbitrarily closely by a piecewise-linear, convex function

$$V(b) = \max_{\alpha \in \Gamma} (\alpha \cdot b), \tag{1}$$

where b is a discrete vector representation of a belief and Γ is a finite set of vectors called α*-vectors*. Each α-vector is associated with an action, and the policy can be executed by selecting the action corresponding to the best α-vector at the current belief b. So a policy can be represented by a set Γ of α-vectors. Policy computation, which, in this case, involves the construction of Γ, is usually performed offline.

Given a policy π, the control of the agent's actions is performed online in real time. It consists of two steps executed repeatedly. The first step is policy execution. If the agent's current belief is b, it then takes the action $a = \pi(b)$, according to the given policy π. The second step is belief estimation. After taking an action a and receiving an observation o, the agent updates its belief state:

$$b'(s') = \tau(b, a, o) = \eta Z(s', a, o) \sum_{s \in S} T(s, a, s') b(s), \tag{2}$$

where η is a normalizing constant. The process then repeats.

More information about POMDPs is available in (Kaelbling, Littman, and Cassandra 1998; Thrun, Burgard, and Fox 2005).

POMDP Planning

Overview of point-based POMDP algorithms

Point-based algorithms have been highly successful in computing approximate solutions to POMDPs with a large number of states. They are based on *value iteration* (Russell and Norvig 2003), which is an iterative approach for computing the optimal value function V^*. Value iteration starts with an initial approximation to V^*, represented as a set Γ of α-vectors. It exploits the fact that V^* must satisfy the Bellman equation and performs backup operations on the approximation by iterating on the Bellman equation, until the iteration

Algorithm 1 Point-based POMDP planning

```
1: Insert the initial belief point $b_0$ as the root of the tree $T_{\mathcal{R}}$.
2: Initialize the set $\Gamma$ of $\alpha$-vectors.
3: **repeat**
4:     Sample new belief points and insert them into $T_{\mathcal{R}}$ by repeatedly calling SAMPLE($T_{\mathcal{R}}, \Gamma$).
5:     Choose a subset of nodes from $T_{\mathcal{R}}$. For each chosen node $b$, BACKUP($T_{\mathcal{R}}, \Gamma, b$).
6:     PRUNE($T_{\mathcal{R}}, \Gamma$).
7: **until** termination conditions are satisfied.
8: **return** $\Gamma$.

SAMPLE($T_{\mathcal{R}}, \Gamma$)
10: Pick a node $b$ from $T_{\mathcal{R}}$, $a \in A$, and $o \in O$.
11: $b' \leftarrow \tau(b, a, o)$.
12: Insert $b'$ into $T_{\mathcal{R}}$ as a child of $b$.

BACKUP($T_{\mathcal{R}}, \Gamma, b$)
13: For all $a \in A$, $o \in O$, $\alpha_{a,o} \leftarrow \text{argmax}_{\alpha \in \Gamma}(\alpha \cdot \tau(b, a, o))$.
14: For all $a \in A, s \in S$, $\alpha_a(s) \leftarrow R(s, a)+$
        $\gamma \sum_{o \in O, s' \in S} T(s, a, s') Z(s', a, o) \alpha_{a,o}(s')$.
15: $\alpha' \leftarrow \text{argmax}_{a \in A}(\alpha_a \cdot b)$
16: Insert $\alpha'$ into $\Gamma$.
```

converges. One key idea behind the success of point-based algorithms is to sample a set of points from the belief space $\mathcal{B}$ and use it as an approximate representation of $\mathcal{B}$, rather than represent $\mathcal{B}$ exactly. The various existing point-based algorithms differ mainly in how they sample $\mathcal{B}$.

Algorithm 1 shows the common structure shared by some of the fastest point-based POMDP algorithms, such as HSVI2 and SARSOP. After initialization, the algorithm iterates over three main steps: SAMPLE, BACKUP, and PRUNE. Let $\mathcal{R} \subseteq \mathcal{B}$ be the set of points reachable from a given initial belief point $b_0 \in \mathcal{B}$ under arbitrary sequences of actions and observations. Most of the recent point-based POMDP algorithms sample from $\mathcal{R}$ instead of $\mathcal{B}$ for computational efficiency. The sampled points form a belief tree $T_{\mathcal{R}}$ (Figure 1). Each node of $T_{\mathcal{R}}$ represents a sampled point $b \in \mathcal{B}$. The root of $T_{\mathcal{R}}$ is the initial belief point b_0. To sample a new point b', we pick a node b from $T_{\mathcal{R}}$ as well as an action $a \in A$ and an observation $o \in O$ according to suitable probability distributions or heuristics. We then compute $b' = \tau(b, a, o)$ using (2) and insert b' into $T_{\mathcal{R}}$ as a child of b. Clearly every point b' obtained this way is reachable from b_0. We then perform backup operations at each node along the path from b' to the root b_0. A backup operation at a node b collates the information in the children of b and propagates it back to b. Invocation of SAMPLE and BACKUP generates new sampled points and α-vectors. However, not all of the α-vectors are useful for constructing an optimal policy and are pruned from Γ to improve computational efficiency.

Algorithm 1 clearly shows the effects of the two "curses" on computational efficiency. In a POMDP with discrete state space, a belief b is typically represented as a vector, and $b(s)$ gives the probability of a robot in state s. The efficiency of all the primitive operations, such as belief update (line 11) and backup (lines 13-15), depends directly on the dimensionality of $\mathcal{B}$, *i.e.*, the number of states for the robot.

Furthermore, if a task requires an effective planning

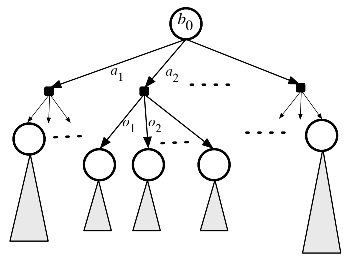


Figure 1: The belief tree $T_{\mathcal{R}}$ rooted at b_0

horizon of h actions and observations, $T_{\mathcal{R}}$ may contain $\mathcal{O}((|A||O|)^h)$ nodes in the worst case, where $|A|$ is the number of actions and $|O|$ is the number of observations. So the size of $T_{\mathcal{R}}$ grows exponentially with h. In principle, deterministic planning suffers from long planning horizons in a similar way. However, each node of $T_{\mathcal{R}}$ corresponds to a point in $\mathcal{B}$. The enormous size of $\mathcal{B}$ fails to place any limit on the growth of $T_{\mathcal{R}}$, and thus aggravates the difficulty.

Reducing the dimensionality of belief space

A high-dimensional belief space arises because a robot's state is not fully observable. Complex robotics systems, however, often have *mixed observability*: even when a robot's state is not fully observable, some components of the state may still be fully observable. For example, consider a mobile robot operating in a multi-storey building. The robot's state consists of its current horizontal position p, orientation θ, and floor ℓ. Suppose that the building has 10 floors and after discretization, the robot may assume any of 100 possible positions on a 10×10 grid in the horizontal plane and 36 orientations. The robot is equipped with a compass, but the geographic positioning system (GPS) is ineffective indoors. Since the robot cannot localize accurately in the horizontal plane, the state (p, θ, ℓ) is not fully observable, and the resulting belief space is 36,000-dimensional. Note, however, that θ is in fact fully observable, if the compass is sufficiently accurate. We may also reasonably assume that ℓ is fully observable. So we only need to maintain a belief on the robot's uncertain horizontal position p. The belief space then becomes a union of 360 disjoint 100-dimensional subspaces. Each subspace corresponds to a specific value of θ, a specific value of ℓ, and beliefs on p. These 100-dimensional subspaces are still large, but a drastic reduction from the original 36,000-dimensional space.

More generally, we propose that the state of a system with mixed observability be factored into two parts (Figure 2). The fully observable state components are represented as a single state variable x, while the partially observable components are represented as another state variable y. Together (x, y) specifies the complete system state, and the state space is factored as $S = X \times Y$, where X is the space of all possible values for x and Y is the space of all possible values for y. In the example above, x represents the orientation θ and the floor ℓ, and y represents the horizontal position p.

The computational advantages of this new factored rep-

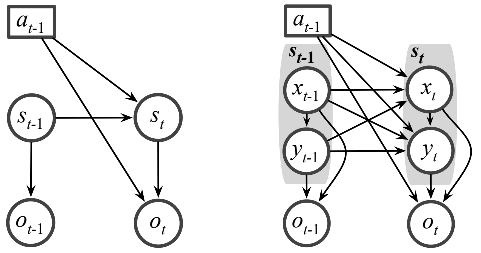


Figure 2: The POMDP model (left) and the MOMDP model (right). A MOMDP state s is factored into two variables: $s = (x, y)$, where x is fully observable and y is partially observable.

resentation become apparent when we consider the belief space $\mathcal{B}$. Since the state variable x is fully observable and known exactly, we only need to maintain a belief b_Y, a probability distribution on the state variable y. Any belief $b \in \mathcal{B}$ on the complete system state $s = (x, y)$ is then represented as (x, b_Y). Let $\mathcal{B}_Y$ denote the space of all beliefs on y. We now associate with each value x of the fully observable state variable a belief space for y: $\mathcal{B}_Y(x) = \{(x, b_Y) \mid b_Y \in \mathcal{B}_Y\}$. $\mathcal{B}_Y(x)$ is a subspace in $\mathcal{B}$, and $\mathcal{B}$ is a union of these subspaces: $\mathcal{B} = \bigcup_{x \in X} \mathcal{B}_Y(x)$. While $\mathcal{B}$ has apparently $|X||Y|$ dimensions, where $|X|$ and $|Y|$ are the number of states in X and Y, each $\mathcal{B}_Y(x)$ has only $|Y|$ dimensions. Effectively we represent the high-dimensional space $\mathcal{B}$ as a union of lower-dimensional subspaces.

Using this new representation, we can redefine the transition function T in the POMDP model and formally introduce a new model called *mixed observability Markov decision process* (MOMDP), which is equivalent to the POMDP model, but more compact. We can perform all the primitive operations in Algorithm 1, such as belief update and backup, in this new model. When the uncertainty in a system is small, specifically, when $|Y|$ is small, the new representation results in dramatic improvement in computational efficiency, due to the reduced dimensionality of the belief space representation. Further details of the point-based MOMDP algorithm can be found in (Ong et al. 2009).

Reducing the time horizon

As we have seen in the overview of point-based POMDP algorithms, the main difficulty here is the exponential growth of the belief tree $T_{\mathcal{R}}$. To address this issue, our basic idea is to sample $\mathcal{R}$ at multiple resolutions by progressively refining $T_{\mathcal{R}}$ and avoid constructing $T_{\mathcal{R}}$ completely unless needed.

Recall that we create a new node b' from an existing node b in $T_{\mathcal{R}}$ using a single action-observation pair: $b' = \tau(b, a, o)$. Instead, we can use a sequence of action-observation pairs $(a_1, o_1, a_2, o_2, \ldots, a_\ell, o_\ell)$ such that $b_1 = \tau(b, a_1, o_1)$ and $b_i = \tau(b_{i-1}, a_i, o_i)$ for $2 \le i \le \ell$. We then set $b' = b_\ell$. This avoids potentially unnecessary branching out of $T_{\mathcal{R}}$ at the intermediate beliefs $b_1, b_2, \ldots, b_{\ell-1}$ and results in a less “bushy” belief tree. The intuitive justification is that many action-observation sequences have similar effects on the belief and exploring one of the sequences is sufficient.

To implement this idea, we build a *roadmap* G, which is a directed graph in a robot's *state space* S. The nodes of G, called *milestones*, are states sampled from S. A directed edge e from a node s to s' is annotated with an action sequence $(a_1, a_2, \ldots, a_\ell)$ that brings the robot from s to s'. The edge e is also annotated with a sampled observation sequence $(o_1, o_2, \ldots, o_\ell)$ that the robot may encounter while executing the actions $(a_1, a_2, \ldots, a_\ell)$. If we treat G as a collection of edges, each representing a sequence of actions and observations, we can then use these edges to construct $T_{\mathcal{R}}$. To generate a new child of a node b in $T_{\mathcal{R}}$, we first associate b with a suitable node s in G with $b(s) > 0$. We then choose an edge e of G with start node s and apply the associated action-observation sequence to b and derive the child node b'. Suppose, for example, that G has maximum degree d and each edge of G contains an action sequence of length ℓ. Then, for a POMDP with time horizon h, $T_{\mathcal{R}}$ contains at most $\mathcal{O}(d^{h/\ell})$ nodes. Comparing this to the worst-case $\mathcal{O}((|A||O|)^h)$ nodes of the usual point-based algorithms indicates that the action sequences encoded in G reduce the effect of long planning horizons, assuming that the size of d can be controlled during the roadmap construction. Since $T_{\mathcal{R}}$ grows exponentially with h, the reduction is significant.

Essentially, the algorithm we propose, *Milestone Guided Sampling* (MiGS), constructs a sampled representation of the state space in the form of a roadmap G, and uses this to guide sampling in $\mathcal{R}$. Now, to sample $\mathcal{R}$ at multiple resolutions, we start with a roadmap G with a small number of nodes. The edges connecting these nodes are necessarily annotated with action-observation sequences that are relatively long. In other words, ℓ is large, which significantly reduces the effective planning horizon, but also results in coarse sampling of $\mathcal{R}$. We then progressively add more nodes to G. This reduces the lengths of action-observation sequences connecting the nodes of G and thus refines the sampling of $\mathcal{R}$. However, as ℓ becomes smaller, the computational costs increase as well.

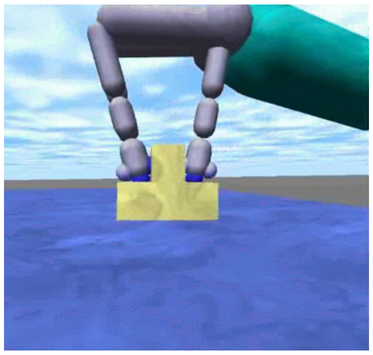
One may question: since G contains only a sampled subset of states in S and not all states, can it possibly lead to a sufficiently good policy? The answer is yes, and we can prove that if our algorithm samples S adequately when constructing G and samples $\mathcal{R}$ adequately when constructing $T_{\mathcal{R}}$, then we can obtain an approximately optimal value function with bounded error. The error bound depends on the resolution at which S and $\mathcal{R}$ are sampled. See (Kurniawati et al. 2009) for the details of roadmap construction and the proof.

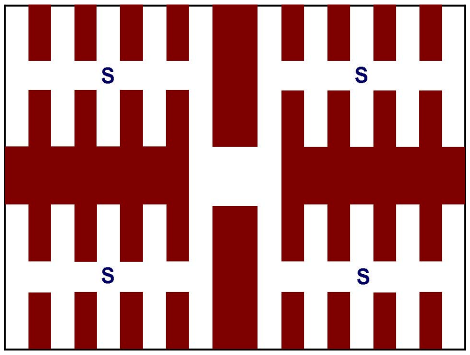
Software Implementation

Based on the above as well as other related ideas, we implemented a C++ software package *Approximate POMDP Planning Library* (APPL), now available for download at `http://motion.comp.nus.edu.sg/projects/pomdp/pomdp.html`.

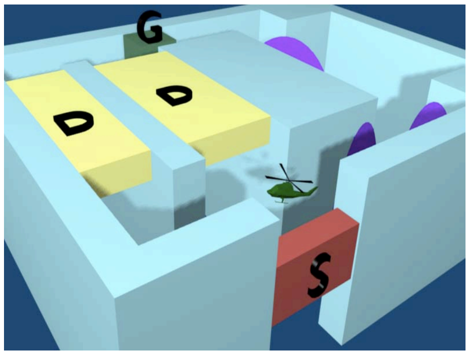
APPL has several useful features for those interested in using POMDPs for robot motion planning:

- APPL provides a new XML-based file format POMDPX for specifying POMDP/MOMDP models in a compact


(*a*)



(*b*)



(*c*)

Figure 3: Three robotic tasks used in the experiments. (*a*) Grasping. (*b*) Target tracking with a sensor network. Dark brown regions indicate obstacles. "S" marks the sensor locations. (*c*) UAV navigation. "S" marks the start region. "G" marks the goal region. "D" marks danger zones. Ellipses indicate landmarks.

and flexible way. APPL is backward compatible and can read the standard POMDP format as well. As an example, the file for RockSample(7,8), a benchmark problem, has 18.9 MB in the standard format, but only 0.1 MB in the new POMDPX format.

- We can also specify the input POMDP/MOMDP models in C++ and access the solver algorithm in APPL through a well-defined programming API.
- APPL provides several ways to examine the computed policy. It contains a simple simulator for estimating the expected total reward of a policy. It can also output a computed policy in a graphical form similar to a finite-state machine controller diagram.

For those interested in improving POMDP algorithms, APPL provides efficient implementation of primitive operations, such as belief update and backup. The belief representation is encapsulated and can be changed without affecting the solver algorithm.

Experiments

To examine the abilities of current POMDP algorithms for robot motion planning, we evaluated our software on robotic tasks with a large number of states or a long time horizon. We chose three distinct robotic tasks reported in the recent literature and modeled them as POMDPs. For each task, we spent a maximum of two hours in computing a policy. The experiments were performed on a PC with a 2.66GHz Intel processor and 2GB memory.

Grasping

In this simplified grasping task (Hsiao, Kaelbling, and Lozano-Pérez 2007), a robot arm with two fingers tries to grasp a stepped block in a two-dimensional environment (Figure 3*a*). Each finger has contact sensors at the tip and the sides. The arm performs compliant guarded moves (left, right, up, and down) and always maintains contact with the surface of the block or the boundary of the environment at the beginning and end of each move. The goal is to move the robot arm and have its two fingers straddle the block so that grasping is possible. Due to the limited sensor information, the uncertainty on the finger positions is high. The robot must perform information-gathering actions in order to grasp the block successfully.

Table 1: Grasping. $|\mathcal{S}| = 1,253$, $|\mathcal{A}| = 6$, and $|\mathcal{O}| = 96$.

	Reward	Time (s)
SARSOP	320.1 ± 0.5	8.0
HSVI2	319.5 ± 0.4	8.0
QMDP	319.9 ± 0.5	<1

The POMDP model of this task is very carefully constructed (Hsiao, Kaelbling, and Lozano-Pérez 2007). It discretizes the geometric environment into a small number of discrete states and uses the guarded moves as actions in order to reduce the planning horizon.

We ran SARSOP (Kurniawati, Hsu, and Lee 2008) and HSVI2 (Smith and Simmons 2005), two of the fastest POMDP algorithms today, on this task. Both algorithms are implemented in C++. SARSOP is included as part of the APPL software package. For HSVI2, we used the newest software released by its original author, zmdp v.1.1.6. We first performed long preliminary runs to determine approximately the reward level for the optimal policies and the amount of time needed to reach it. We then ran the algorithms to reach this level. To estimate the expected total reward of the resulting policy, we performed a sufficiently large number of simulation runs until the variance in the estimated value was small.

The results are shown in Table 1. Column 2 of the table shows the estimated expected total rewards for the computed policies and the 95% confidence intervals. Column 3 shows the running times for policy computation. As a baseline comparison, we also ran QMDP, a popular heuristic uncertainty planning algorithm, which assumes that after one step, the state becomes fully observable. The running time of QMDP gives a rough indication of the difficulty of the uncertainty planning task.

In this task, the challenges of a large state space and a long planning horizon are tackled through careful POMDP modeling. The resulting POMDP model in fact has a small state space and a relatively short planning horizon. Both SARSOP and HSVI2 computed approximately optimal policies quickly. Even the less capable QMDP was able to attain comparable results. However, such clever modeling requires significant insight of the underlying task. In general, it is not easy to come up with a POMDP model with a compact state space and a short planning horizon.

Target tracking with a sensor network

In this target tracking task (O'Kane and Xu 2009), a robot watches over a person, moving around in an office building and attends to his call for help as soon as possible. The environment is modeled as a two-dimensional grid (Figure 3*b*). At each time step, the robot either stays or moves to one of its eight neighboring cells. Moving consumes energy, while staying does not. The target, *i.e.*, the person, moves around and calls the robot for help when needed. He then stays in place until the robot arrives to help him.

To respond to a call for help, the robot needs the person's location. A sparse sensor network is installed in the building. Each node in the network is a one-bit proximity sensor that senses whether a person is within a small immediate neighborhood. The sensors are quite noisy. Both false positives and false negatives may occur. Information from one node propagates to other nodes in the network instantaneously. However, the robot can access this information only if it is close enough to one of the sensors. Although the person's position is uncertain and the robot cannot anticipate beforehand when its help would be required, the robot must come to the person's aid as soon as possible, and at the same time, avoid consuming energy unnecessarily.

In this task, the size of the state space increases rapidly with that of the geometric environment, because the state contains both the robot's and the target's positions. Furthermore, the robot needs to keep track of whether the person is calling for help. So a moderately-sized environment with 124 grid cells, which we used in our experiments, has over $30,000$ states, resulting in a $30,000$-dimensional belief space!

Fortunately, we can represent the belief space much more efficiently using the MOMDP approach described earlier. In this tracking task, we can reasonably assume that the uncertainty on the target's position is much greater than that on the robot's position. If robot localization is accurate enough, we can even assume that the only uncertainty is on the target's position, and that the robot's position is fully observable. We thus model this tracking task as a MOMDP: the fully observable variable x represents the robot's position and the person's call for help, while the partially observable variable y represents the person's position. This greatly reduces the belief space dimensionality. The belief space is now represented as a union of 124-dimensional subspaces rather than a single $30,000$-dimensional space.

We ran the SARSOP algorithm using the MOMDP model for this task. We also ran HSVI2, but using the standard POMDP model, because the zmdp software does not support the MOMDP representation.

Table 2: Target tracking with a sensor network.

	Reward	Time (s)
124-cell grid		
$\lvert\mathcal{S}\rvert$=30,752, $\lvert\mathcal{A}\rvert$=9, $\lvert\mathcal{O}\rvert$=20,088		
$\lvert\mathcal{X}\rvert$=248, $\lvert\mathcal{Y}\rvert$=124		
MOMDP	2.1 ± 0.6	1232
HSVI2	–	–
QMDP	–	–
38-cell grid		
$\lvert\mathcal{S}\rvert$=2,888, $\lvert\mathcal{A}\rvert$=9, $\lvert\mathcal{O}\rvert$=684		
$\lvert\mathcal{X}\rvert$=76, $\lvert\mathcal{Y}\rvert$=38		
MOMDP	29.1 ± 1.9	60
HSVI2	27.2 ± 6.0	3730
QMDP	10.0 ± 1.3	2

Results were obtained using a similar procedure as that in the previous experiment, and are shown in Table 2. Using the MOMDP representation, SARSOP computed a reasonable policy in about 20 minutes. HSVI2 and QMDP use the standard POMDP models and failed to load the model files, because they are too large.

To make a comparison, we constructed a coarser grid of 38 cells for the same geometric environment and repeated the experiments with the three algorithms. The results clearly demonstrate the benefit of the MOMDP model over the standard POMDP model. Using the MOMDP model, SARSOP achieved the best reward level after only one minute of computation. HSVI2 reached a similar reward level after one hour. QMDP was unable to achieve a comparable reward level at all.

UAV navigation

In this navigation task (He, Prentice, and Roy 2008), an unmanned aerial vehicle (UAV) navigates in an indoor environment where GPS signal is unavailable. The environment is modeled as a three-dimensional grid with 5 levels and 18×14 positions on each level (Figure 3*c*). The robot's state consists of its grid position and the (discretized) pitch and yaw angles. The robot needs to navigate from a start region to a goal region. At each time step, the robot can either rotate in place or move forward to one of its adjacent cells according to its heading. However, when the robot tries to move forward, it may drift to its left or right, or remain at the same cell. Moreover, there are danger zones that the robot must avoid.

As there is no GPS signal, the robot must rely on landmarks in the environment to localize. Due to the limited sensing ability, the robot can observe a landmark only when it is close enough and is headed towards the landmark. Despite poor localization, the robot must get to the goal region as quickly as possible while avoiding obstacles and dangerous zones along the way.

For most robot navigation tasks, a major difficulty is the long planning horizon. Here, this difficulty is aggravated by the relatively large state space. To examine how effective the MiGS algorithm is in managing long planning horizons, we

Table 3: UAV navigation. $|\mathcal{S}| = 16,969, |\mathcal{A}| = 5$, and $|\mathcal{O}| = 14$.

	Success Rate (%)	Reward	Time (s)
MiGS	88.2	371.1 ± 67.9	1000
HSVI2	24.0	65.6 ± 8.3	7201
QMDP	0.1	0.7 ± 0.2	<1

applied it to this task and compared the results with those from HSVI2, using the same procedure as in the previous experiments, and running both algorithms for a maximum of two hours. The MiGS algorithm is implemented in C++, but it is yet to be integrated into the APPL software package.

The results are shown in Table 3. In addition to the reward levels and running times, we also show the success rates, *i.e.*, the percentage of simulation runs in which the UAV reaches the goal. The results indicate that MiGS is effective in alleviating the difficulty of long planning horizons. It achieved a success rate of 88.2% after around 15 minutes of policy computation. In comparison, HSVI2 had a success rate of 24.0% after 2 hours.

HSVI2 had poor performance on this task because its search heuristic failed to explore the reachable belief space $\mathcal{R}$ adequately, due to the sheer size of $\mathcal{R}$ in this long horizon task. MiGS uses state space information to sample $\mathcal{R}$ at multiple resolutions and thus explores a much larger part of $\mathcal{R}$ efficiently.

The results also show that QMDP performed very poorly on this task. QMDP assumes that after one step, the state becomes fully observable. However, in this task, many steps are required before any observations become available. As a result, the policy generated by QMDP seldom brings the UAV to the goal region. This stresses the importance of taking into account uncertainty in a principled and systematic way for complex planning task.

Conclusion

POMDPs provide a principled general framework for motion planning in uncertain and dynamic environments and have been successfully used for various robotic tasks in motion planning (Hsiao, Kaelbling, and Lozano-Pérez 2007; Pineau, Gordon, and Thrun 2003; Smith and Simmons 2005). However, in scaling up to more complex tasks, POMDP planning algorithms are challenged by two obstacles – the "curse of dimensionality" and the "curse of history". We have presented two ideas that help to overcome these two challenges – experimental results show that algorithms based on our approaches significantly outperform the fastest POMDP solvers today. The results also highlight the necessity of taking into account uncertainty in a principled and systematic way, for motion planning in complex tasks.

Ten years ago, the best POMDP algorithm could solve POMDPs with a dozen states. Five years ago, a point-based algorithm solved a POMDP with almost 900 states, and it was a major accomplishment. Nowadays, POMDPs with hundreds of states can often be solved in seconds, and much larger POMDPs can be solved in reasonable time. We hope that our work is a step further in scaling up POMDP algorithms and ultimately making them practical for robot motion planning in uncertain and dynamic environments.

Acknowledgments We thank Lim Zhanwei for his help in generating some of the POMDP models used in this work. This work is supported in part by AcRF grant R-252-000-327-112 from the Ministry of Education of Singapore.

References

Choset, H.; Lynch, K.; Hutchinson, S.; Kantor, G.; Burgard, W.; Kavraki, L.; and Thrun, S. 2005. *Principles of Robot Motion : Theory, Algorithms, and Implementations*. The MIT Press. chapter 7.

He, R.; Prentice, S.; and Roy, N. 2008. Planning in information space for a quadrotor helicopter in a GPS-denied environment. In *Proc. IEEE Int. Conf. on Robotics & Automation*, 1814–1820.

Hsiao, K.; Kaelbling, L.; and Lozano-Pérez, T. 2007. Grasping POMDPs. In *Proc. IEEE Int. Conf. on Robotics & Automation*, 4485–4692.

Hsu, D.; Lee, W.; and Rong, N. 2008. A point-based POMDP planner for target tracking. In *Proc. IEEE Int. Conf. on Robotics & Automation*, 2644–2650.

Kaelbling, L.; Littman, M.; and Cassandra, A. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101(1–2):99–134.

Kurniawati, H.; Du, Y.; Hsu, D.; and Lee, W. 2009. Motion planning under uncertainty for robotic tasks with long time horizons. In *Proc. Int. Symp. on Robotics Research*.

Kurniawati, H.; Hsu, D.; and Lee, W. 2008. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proc. Robotics: Science and Systems*.

Latombe, J. 1991. *Robot Motion Planning*. Boston, MA: Kluwer Academic Publishers.

LaValle, S. 2006. *Planning Algorithms*. Cambridge University Press.

Lozano-Pérez, T. 1983. Spatial planning: A configuration space approach. *IEEE Transactions on Computers* 32(2):108–120.

O'Kane, J., and Xu, W. 2009. Energy-efficient target tracking with a sensorless robot and a network of unreliable one-bit proximity sensors. In *Proc. IEEE Int. Conf. on Robotics & Automation*.

Ong, S.; Png, S.; Hsu, D.; and Lee, W. 2009. POMDPs for robotic tasks with mixed observability. In *Proc. Robotics: Science and Systems*.

Papadimitriou, C., and Tsisiklis, J. 1987. The complexity of Markov decision processes. *Mathematics of Operations Research* 12(3):441–450.

Pineau, J.; Gordon, G.; and Thrun, S. 2003. Point-based value iteration: An anytime algorithm for POMDPs. In *Proc. Int. Jnt. Conf. on Artificial Intelligence*, 477–484.

Pineau, J.; Gordon, G.; and Thrun, S. 2005. Point-based approximations for fast POMDP solving. In *Proc. Int. Symp. on Robotics Research*.

Roy, N.; Gordon, G.; and Thrun, S. 2005. Finding approximate POMDP solutions through belief compression. *J. Artificial Intelligence Research* 23:1–40.

Russell, S., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Smallwood, R., and Sondik, E. 1973. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research* 21:1071–1088.

Smith, T., and Simmons, R. 2005. Point-based POMDP algorithms: Improved analysis and implementation. In *Proc. Uncertainty in Artificial Intelligence*.

Spaan, M., and Vlassis, N. 2005. Perseus: Randomized point-based value iteration for POMDPs. *J. Artificial Intelligence Research* 24:195–220.

Thrun, S.; Burgard, W.; and Fox, D. 2005. *Probabilistic Robotics*. The MIT Press.