Robotics: Science and Systems 2009
Seattle, WA, USA, June 28 - July 1, 2009

POMDPs for Robotic Tasks with Mixed Observability

Sylvie C.W. Ong Shao Wei Png David Hsu Wee Sun Lee
Department of Computer Science, National University of Singapore
Singapore 117590, Singapore

***Abstract*— Partially observable Markov decision processes (POMDPs) provide a principled mathematical framework for motion planning of autonomous robots in uncertain and dynamic environments. They have been successfully applied to various robotic tasks, but a major challenge is to scale up POMDP algorithms for more complex robotic systems. Robotic systems often have *mixed observability*: even when a robot's state is not fully observable, some components of the state may still be fully observable. Exploiting this, we use a factored model to represent separately the fully and partially observable components of a robot's state and derive a compact lower-dimensional representation of its belief space. We then use this factored representation in conjunction with a point-based algorithm to compute approximate POMDP solutions. Separating fully and partially observable state components using a factored model opens up several opportunities to improve the efficiency of point-based POMDP algorithms. Experiments show that on standard test problems, our new algorithm is many times faster than a leading point-based POMDP algorithm.**

I. Introduction

Planning under uncertainty is a critical ability for autonomous robots operating in uncontrolled environments, such as homes or offices. In robot motion planning, uncertainty arises from two main sources: a robot's action and its perception. If the effect of a robot's action is uncertain, but its state is fully observable, then Markov decision processes (MDPs) provide an adequate model for planning. MDPs with a large number of states can often be solved efficiently (see, *e.g.*, [2]). When a robot's state is not fully observable, maybe due to noisy sensors, partially observable Markov decision processes (POMDPs) become necessary, and solving POMDPs is much more difficult. Despite the impressive progress of point-based POMDP algorithms in recent years [9], [11], [16], [19], [20], solving POMDPs with a large number of states remains a challenge. It is, however, important to note that robotic systems 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 equipped with an accurate compass, but not a geographic positioning system (GPS). Its orientation is fully observable, though its position may only be partially observable. We refer to such problems as *mixed observability MDPs* (MOMDPs), a special class of POMDPs. In this work, we separate the fully and partially observable components of the state through a factored model and show that the new representation drastically improves the speed of POMDP planning, leading to a much faster algorithm for robotic systems with mixed observability.

In a POMDP, a robot's state is not fully observable. Thus we model it as a *belief*, which is a probability distribution over all possible robot states. The set of all beliefs form the *belief space* $\mathcal{B}$. The concept of belief space is similar to that of configuration space, except that each point in $\mathcal{B}$ represents a probability distribution over robot states rather than a single robot configuration or state. Intuitively, the difficulty of solving POMDPs is due to the "curse of dimensionality": in a POMDP with discrete states, the belief space $\mathcal{B}$ has dimensionality equal to $|\mathcal{S}|$, the number of robot states. The size of $\mathcal{B}$ thus grows exponentially with $|\mathcal{S}|$. Consider, for example, the navigation problem for an autonomous underwater vehicle (AUV). The state of the robot vehicle consists of its 3-D position and orientation. Suppose that after discretization, the robot may assume any of 100 possible positions on a 10×10 grid in the horizontal plane, 5 depth levels, and 24 orientations. The resulting belief space is 12,000-dimensional!

Now if the robot has an accurate pressure sensor and a gyroscope, we may reasonably assume that the depth level and the orientation are known exactly and fully observable, and only maintain a belief on the robot's uncertain horizontal position. In this case, the belief space becomes a union of 120 disjoint 100-dimensional subspaces. Each subspace corresponds to an exact depth level, an exact orientation, and beliefs on the uncertain horizontal positions. These 100-dimensional subspaces are still large, but a substantial reduction from the original 12,000-dimensional space.

The main idea of our approach is to exploit full observability whenever possible to gain computational efficiency. We separate the fully and partially observable state components using a factored model and represent explicitly the disjoint belief subspaces in a MOMDP so that all operations can be performed in these lower-dimensional subspaces.

The observability of a robot's state is closely related to sensor limitations. Two common types of sensor limitations are addressed in this work. In the first case, some components of a robot's state are sensed accurately and considered fully observable. Our approach handles this by separating the state components with high sensing precision from the rest. The second case is more subtle. Some sensors have *bounded* errors, but are not accurate enough to allow any assumption of fully observable state components *a priori*. Nevertheless we show that a robot with such sensors can be modeled as a MOMDP by (re)parameterizing the robot's state space. The reparameterization technique enables a much broader class of planning problems to benefit from our approach.

We tested our algorithm on three distinct robotic tasks with large state spaces. The results show that it significantly outperforms a leading point-based POMDP algorithm.

II. Background

A. POMDPs

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

In each time step, the agent lies in some state $s \in \mathcal{S}$. It takes some action $a \in \mathcal{A}$ and moves from s to a new state s'. Due to the uncertainty in action, the end state s' is modeled 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 to gather information on its own state. Due to the uncertainty in observation, the observation result $o \in \mathcal{O}$ is again modeled as a conditional probability function $Z(s, a, o) = p(o|s, a)$. See Fig. 1 for an illustration.

To elicit desirable agent behavior, we define a suitable reward function $R(s, a)$. In each time step, the agent receives a reward $R(s, a)$ if it takes action a in state s. The agent's goal is to maximize its expected total reward by choosing a suitable sequence of actions. For infinite-horizon POMDPs, the sequence of actions has infinite length. We 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 $\mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$, where s_t and a_t denote the agent's state and action at time t, respectively.

For a POMDP, planning means computing an *optimal policy* that maximizes the expected total reward. Normally, a policy is a mapping from the agent's state to an action. It tells the agent what action to take in each state. However, in a POMDP, the agent's state is partially observable and not known exactly. We rely on the concept of a belief, which is a probability distribution over $\mathcal{S}$. A POMDP policy $\pi: \mathcal{B} \rightarrow \mathcal{A}$ maps a belief $b \in \mathcal{B}$ to a prescribed action $a \in \mathcal{A}$.

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

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

where Γ is a finite set of vectors called α-vectors, b is the discrete vector representation of a belief, and $\alpha \cdot b$ denotes the inner product of an α-vector and b. Each α-vector is associated with an action. The policy can be executed by evaluating (1) to find the action corresponding to the best α-vector at the current belief. So a policy can be represented by a value function $V(b)$ consisting of a set Γ of α-vectors. Policy computation, which, in this case, involves the construction of Γ, is usually performed offline.

Given a policy, represented as a value function $V(b)$, the control of the agent's actions, also called policy execution, is performed online in real time. It consists of two steps executed repeatedly. The first step is action selection. If the agent's current belief is b, it finds the action a that maximizes $V(b)$ by evaluating (1). The second step is belief estimation. After the agent takes an action a and receives an observation o, its new belief b' is given by

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

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

B. Related Work

POMDPs are a powerful framework for planning under uncertainty [8], [17]. It has a solid mathematical foundation and wide applicability. Its main disadvantage is high computational complexity [10]. As mentioned in Section I, the belief space $\mathcal{B}$ used for POMDP policy computation grows exponentially with the number of states that an agent has. The resulting curse of dimensionality is one major obstacle to efficient solution of POMDPs. There are several approaches to overcome the difficulty, including sampling $\mathcal{B}$, building factored models of $\mathcal{B}$ [1], or building lower-dimensional approximations of $\mathcal{B}$ [13], [14]. In recent years, point-based algorithms, which are based on the idea of sampling, have made impressive progress in computing approximate solutions to large POMDPs [9], [11], [16], [19], [20]. They have been successfully applied to a variety of non-trivial robotic tasks, including coastal navigation, grasping, target tracking, and exploration [5], [11], [12], [19]. In some cases, POMDPs with hundreds of states have been solved in a matter of seconds (see, *e.g.*, [7], [16], [19]).

Our work aims at overcoming the difficulty of high-dimensional belief spaces and further scaling up point-based POMDP algorithms for realistic robotic tasks. The main idea is to use a factored model to represent separately the fully and partially observable components of an agent's state and derive a compact lower-dimensional representation of $\mathcal{B}$. We then use this new representation in conjunction with a point-based algorithm to compute approximate POMDP solutions. A related idea has been used in medical therapy planning [4].

III. Mixed Observability MDPs

A. The MOMDP Model

In the standard POMDP model, the state lumps together multiple components. Consider again our AUV navigation example from Section I. The robot's state consists of its horizontal position, depth, and orientation. In contrast, a factored POMDP model separates the multiple state components and represents each as a distinct state variable. If three variables p, d, and θ represent the AUV's horizontal position, depth, and orientation, respectively, then the state space is a cross product of three subspaces: $\mathcal{S} = \mathcal{S}_p \times \mathcal{S}_d \times \mathcal{S}_\theta$. This allows for a more structured and compact representation of transition, observation, and reward functions in a POMDP.

We propose to represent a robotic system with mixed observability as a factored POMDP, specifically, a factored POMDP model with mixed state variables. We call our

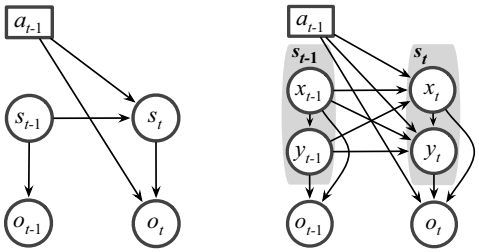

Fig. 1. The standard POMDP model (left) and the MOMDP model (right). A MOMDP state s is factored into two variables: $s = (x, y)$, where x is fully observable and y is partially observable.

model a MOMDP. In a MOMDP, 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. Thus (x, y) specifies the complete system state, and the state space is factored as $\mathcal{S} = \mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ is the space of all possible values for x and $\mathcal{Y}$ is the space of all possible values for y. In our AUV example, x represents the depth and the orientation (d, θ), and y represents the horizontal position p.

Formally a MOMDP model is specified as a tuple $(\mathcal{X}, \mathcal{Y}, \mathcal{A}, \mathcal{O}, T_x, T_y, Z, R, \gamma)$. The conditional probability function $T_x(x, y, a, x') = p(x'|x, y, a)$ gives the probability that the fully observable state variable has value x' if the robot takes action a in state (x, y), and $T_y(x, y, a, x', y') = p(y'|x, y, a, x')$ gives the probability that the partially observable state variable has value y' if the robot takes action a in state (x, y) and the fully observable state variable has value x'. Compared with the standard POMDP model, the MOMDP model uses a factored state-space representation $\mathcal{X} \times \mathcal{Y}$, with the corresponding probabilistic state-transition functions T_x and T_y. All other aspects remain the same. See Fig. 1 for a comparison.*

So far, the changes introduced by the MOMDP model seem mostly notational. The computational advantages 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: $\mathcal{B} = \bigcup_{x \in \mathcal{X}} \mathcal{B}_y(x)$. Observe that while $\mathcal{B}$ has $|\mathcal{X}||\mathcal{Y}|$ dimensions, where $|\mathcal{X}|$ and $|\mathcal{Y}|$ are the number of states in $\mathcal{X}$ and $\mathcal{Y}$, each $\mathcal{B}_y(x)$ has only $|\mathcal{Y}|$ dimensions. Effectively we represent the high-dimensional space $\mathcal{B}$ as a union of lower-dimensional subspaces. When the uncertainty in a system is small, specifically, when $|\mathcal{Y}|$ is small, the MOMDP model leads to dramatic improvement in computational efficiency, due to the reduced dimensionality of the space.

Now consider how we would represent and execute a MOMDP policy. As mentioned in Section II-A, a POMDP policy can be represented as a value function $V(b) = \max_{\alpha \in \Gamma}(\alpha \cdot b)$, where Γ is a set of α-vectors. In a MOMDP, a belief is given by (x, b_y), and the belief space $\mathcal{B}$ is union of subspaces $\mathcal{B}_y(x)$ for $x \in \mathcal{X}$. Correspondingly, a MOMDP value function $V(x, b_y)$ is represented as a collection of α-vector sets: $\{\Gamma_y(x) \mid x \in \mathcal{X}\}$, where for each x, $\Gamma_y(x)$ is a set of α-vectors defined over $\mathcal{B}_y(x)$. To evaluate $V(x, b_y)$, we first find the right α-vector set $\Gamma_y(x)$ using the x value and then find the maximum α-vector from the set:

$$V(x, b_y) = \max_{\alpha \in \Gamma_y(x)}(\alpha \cdot b_y). \tag{3}$$

In general, any value function $V(b) = \max_{\alpha \in \Gamma}(\alpha \cdot b)$ can be represented in this new form, as stated in the theorem below.

Theorem 1: Let $\mathcal{B} = \bigcup_{x \in \mathcal{X}} \mathcal{B}_y(x)$ be the belief space of a MOMDP with state space $\mathcal{X} \times \mathcal{Y}$. If $V(b) = \max_{\alpha \in \Gamma}(\alpha \cdot b)$ is any value function over $\mathcal{B}$ in the standard POMDP form, then $V(b)$ is equivalent to a MOMDP value function $V'(x, b_y) = \max_{\alpha \in \Gamma_y(x)}(\alpha \cdot b_y)$ such that for any $b = (x, b_y)$ with $b \in \mathcal{B}$, $x \in \mathcal{X}$, and $b_y \in \mathcal{B}_y(x)$, $V(b) = V'(x, b_y)$.†.

Geometrically, each α-vector set $\Gamma_y(x)$ represents a restriction of the POMDP value function $V(b)$ to the subspace $\mathcal{B}_y(x)$: $V_x(b_y) = \max_{\alpha \in \Gamma_y(x)}(\alpha \cdot b_y)$. In a MOMDP, we compute only these lower-dimensional restrictions $\{V_x(b_y) \mid x \in \mathcal{X}\}$, because $\mathcal{B}$ is simply a union of subspaces $\mathcal{B}_y(x)$ for $x \in \mathcal{X}$.

A comparison of (1) and (3) also indicates that (3) often results in faster policy execution, because action selection can be performed more efficiently. First, each α-vector in $\Gamma_y(x)$ has length $|\mathcal{Y}|$, while each α-vector in Γ has length $|\mathcal{X}||\mathcal{Y}|$. Furthermore, in a MOMDP value function, the α-vectors are partitioned into groups according to the value of x. We only need to calculate the maximum over $\Gamma_y(x)$, which is potentially much smaller than Γ in size.

In summary, by factoring out the fully and partially observable state variables, a MOMDP model reveals the internal structure of the belief space as a union of lower-dimensional subspaces. We want to exploit this structure and perform all operations on beliefs and value functions in these lower-dimensional subspaces rather than the original belief space. Before we describe the details of our algorithm, let us first look at how MOMDPs can be used to handle uncertainty commonly encountered in robotic systems.

B. Modeling Robotic Tasks with MOMDPs

Sensor limitations are a major source of uncertainty in robotic systems and are closely related to observability. If a robot's state consists of several components, some are fully observable, possibly due to accurate sensing, but others are not. This is a natural case for modeling with MOMDPs. All fully observable components are grouped together and modeled by the variable x. The other components are modeled by the variable y.

*A MOMDP can be regarded as an instance of dynamic Bayesian network (DBN). Following the DBN methodology, we could factor x or y further, but this may lead to difficulty in value function representation.

†Due to space limitations, the proofs of all the theorems are provided in the full version of the paper, which will be available on-line at `http://motion.comp.nus.edu.sg/papers/rss09.pdf`

Sometimes, however, a system does not appear to have mixed observability: none of the sensed state components is fully observable. Is it still possible to model it as a MOMDP? The answer is yes under certain conditions, despite the absence of obvious fully observable state components.

1) Pseudo Full Observability: All sensors are ultimately limited in resolution. It is task-dependent to decide whether the sensor resolution is accurate enough to make the sensed state component fully observable. For example, a robot searches for an unseen target and has small uncertainty on its own position. It is reasonable to assume that the robot position is fully observable, as the uncertainty on the robot position is small compared to that on the target position and the robot's behavior depends mostly on the latter. By treating the robot position as fully observable, we can take advantage of the MOMDP model for faster policy computation and execution. Note, however, that treating the unseen target's position as fully observable is not reasonable and unlikely to lead to a useful policy.

For increased robustness, we can actually execute a computed MOMDP policy on the corresponding POMDP model, which does *not* assume any fully observable state variables. The POMDP treats both state variables x and y as partially observable and maintains a belief b over them. To account for the additional uncertainty on x in the POMDP, our idea is to define a new value function $V_{\text{P}}(b)$ by averaging over the value function $V(x, b_{\mathcal{Y}})$, which represents the computed MOMDP policy. For this, we first calculate a belief $b_{\mathcal{X}}$ on x by marginalizing out y: $b_{\mathcal{X}}(x) = \sum_{y \in \mathcal{Y}} b(x, y)$. We then calculate the belief $b_{\mathcal{Y}|x}$ on y, conditioned on the x value: $b_{\mathcal{Y}|x}(y) = b(x, y)/b_{\mathcal{X}}(x)$. Now the new value function

$$V_{\text{P}}(b) = \sum_{x \in \mathcal{X}} b_{\mathcal{X}}(x) V(x, b_{\mathcal{Y}|x})$$

can be used to generate a policy for the POMDP through one-step look-ahead search.

Let $V^*(b)$ and $V^*(x, b_{\mathcal{Y}})$ be respectively the optimal value functions for a POMDP and the corresponding MOMDP under the assumption of fully observable state variables. It can be shown that the value function $V_{\text{P}}(b)$ constructed from $V^*(x, b_{\mathcal{Y}})$ is an upper bound on $V^*(b)$. In this sense, the MOMDP model is an approximation to the POMDP model. The MOMDP model is less accurate due to the additional assumption, but has substantial computational advantages.

2) Reparameterized Full Observability: Sometimes sensors are not accurate enough to justify an assumption of full observability for any sensed component. However, we can still model such a system as a MOMDP by reparameterizing the state space $\mathcal{S}$. For example, in a navigation task, the position sensor is noisy, but accurate enough to localize the robot to a small region around the actual position. Define $h(o)$, the *preimage* of an observation o, to be the set of states that have non-zero probability of emitting o. We say that a system has *bounded uncertainty* if the preimage of its observation is always a small subset of $\mathcal{S}$: $\max_{o \in \mathcal{O}}(|h(o)|/|\mathcal{S}|) < c$ for some constant $c \ll 1$. We show below that any system modeled as a POMDP, can be equivalently constructed as an MOMDP by reparameterizing $\mathcal{S}$, even if none of the sensed state components are fully observable.

We first illustrate the reparameterization technique on the robot navigation task, modeled as a standard POMDP $(\mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R, \gamma)$. The state $s \in \mathcal{S}$ indicates the robot position, which is partially observable, due to inaccurate sensing, and $o \in \mathcal{O}$ is the observed robot position. Our goal is to reparameterize $\mathcal{S}$ so that $s = (x, y)$, where x is fully observable and y is partially observable. We choose x to be o, the observed robot position, and define y as the offset of the actual position from the observed position. Using this parameterization, we can construct the new state-transition functions $T_{\mathcal{X}}$ and $T_{\mathcal{Y}}$ from the old state-transition function T and observation function Z:

$$T_{\mathcal{X}}(x, y, a, x') = \sum_{\sigma \in \mathcal{S}} T(s, a, \sigma) Z(\sigma, a, x'), \tag{4}$$

$$T_{\mathcal{Y}}(x, y, a, x', y') = T(s, a, s') Z(s', a, x') / T_{\mathcal{X}}(x, y, a, x'), \tag{5}$$

where $s = (x, y)$ and $s' = (x', y')$. The correctness of this construction can be verified by applying the definitions and the basic probability rules. Similarly, we construct the new observation function Z_{M} and the new reward function R_{M}: $Z_{\text{M}}(x, y, a, o) = 1$ if and only if $x = o$, and $R_{\text{M}}(x, y, a) = R(s, a)$, where $s = (x, y)$.

In the general case, we use x to index the preimage of each observation $o \in \mathcal{O}$ and use y to indicate the exact state within the preimage. The resulting reparameterized MOMDP has the following property:

Theorem 2: The POMDP $(\mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R, \gamma)$ and the reparameterized MOMDP $(\mathcal{X}, \mathcal{Y}, \mathcal{A}, \mathcal{O}, T_{\mathcal{X}}, T_{\mathcal{Y}}, Z_{\text{M}}, R_{\text{M}}, \gamma)$ with $\mathcal{X} = \mathcal{O}$ are equivalent: Let b and b' be the beliefs reached after an arbitrary sequence of actions and observations, $a_1, o_1, \cdots, a_t, o_t$, by the POMDP and MOMDP models respectively. Then b and b' represent the same belief in the original POMDP state space. The probability of observation given the history, $p(o_t | a_1, o_1, \cdots, a_t)$, and the expected total reward for any sequence of actions and observations, is the same in both models. Consequently, any policy has the same expected reward for both models.

The reparameterization can be performed on any POMDP. The resulting MOMDP brings computational advantages if $|\mathcal{Y}|$ is small. This happens when a system has bounded uncertainty with a small constant c, meaning that the observations are informative. Furthermore, we can relax the condition of bounded uncertainty and require that the preimages are bounded for some, rather than all state components. The reparameterization is beneficial in this general case as well.

IV. Computing MOMDP Policies

A. Overview

For policy computation, we combine the MOMDP model with a point-based POMDP algorithm. Point-based algorithms have been highly successful in computing approximate solutions to large POMDPs. Their key idea is to sample a set of points from $\mathcal{B}$ and use it as an approximate representation of $\mathcal{B}$, rather than represent $\mathcal{B}$ exactly. They

Algorithm 1 Point-based MOMDP policy computation.

1: Initialize the α-vectors, $\Gamma = \{\Gamma_{\mathcal{Y}}(x) \mid x \in \mathcal{X}\}$, representing the lower bound $\underline{V}$ on the optimal value function V^*. Initialize the upper bound $\overline{V}$ on V^*.
2: Insert the initial belief point $(x_0, b_{\mathcal{Y}0})$ as the root of the tree $T_{\mathcal{R}}$.
3: **repeat**
4: SAMPLE($T_{\mathcal{R}}$, Γ).
5: Choose a subset of nodes from $T_{\mathcal{R}}$. For each chosen node $(x, b_{\mathcal{Y}})$, BACKUP($T_{\mathcal{R}}, \Gamma, (x, b_{\mathcal{Y}})$).
6: PRUNE($T_{\mathcal{R}}$, Γ).
7: **until** termination conditions are satisfied.
8: **return** Γ.

also maintain a set of α-vectors as an approximation to the optimal value function. The various point-based algorithms differ mainly in their strategies for sampling $\mathcal{B}$ and constructing α-vectors.

The MOMDP model enables us to treat a high-dimensional belief space as a union of lower-dimensional subspaces. We represent explicitly these subspaces and compute only the restriction of the value function to these subspaces. The idea is general and is independent of the strategies for belief point sampling or α-vector construction. Hence the MOMDP model can be used in conjunction with any of the existing point-based algorithms. However, to make the presentation concrete, we describe our approach based on the SARSOP algorithm [9], one of the leading point-based POMDP solvers.

Our point-based MOMDP algorithm is based on *value iteration* [15]. Exploiting the fact that the optimal value function V^* must satisfy the Bellman equation, value iteration starts with an initial approximation to V^* and performs backup operations on the approximation by iterating on the Bellman equation until the iteration converges.

In our algorithm, we sample incrementally a set of points from $\mathcal{B}$ and maintain a set of α-vectors, which represents a piecewise-linear lower-bound approximation $\underline{V}$ to V^*. To improve the approximation $\underline{V}$, we perform backup operations on the α-vectors at the sampled points. A backup operation is essentially an iteration of dynamic programming, which improves the approximation by looking ahead one step further. With suitable initialization, $\underline{V}$ is always a lower bound on V^*, and converges to V^* under suitable conditions [6], [11], [19]. The MOMDP model enables the primitive operations in the algorithm, such as α-vector backup and pruning, to be performed more efficiently.

B. The Algorithm

After initialization, our algorithm iterates over three main functions, SAMPLE, BACKUP, and PRUNE (Algorithm 1).

1) SAMPLE*:* Let $\mathcal{R} \subset \mathcal{B}$ be the set of points reachable from a given initial belief point $b_0 = (x_0, b_{\mathcal{Y}0})$ under arbitrary sequences of actions. Most of the recent point-based POMDP algorithms sample from $\mathcal{R}$ instead of $\mathcal{B}$ for computational efficiency. The SARSOP algorithm aims to be even more efficient by focusing the sampling near $\mathcal{R}^*$, the subset of points reachable from $(x_0, b_{\mathcal{Y}0})$ under *optimal* sequences of actions, usually a much smaller space than $\mathcal{R}$.

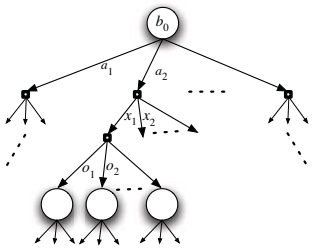

Fig. 2. The belief search tree rooted at $b_0 = (x_0, b_{\mathcal{Y}0})$. Each circle indicates a node representing a MOMDP belief state $(x, b_{\mathcal{Y}})$.

To sample near $\mathcal{R}^*$, we maintain both a lower bound $\underline{V}$ and an upper bound $\overline{V}$ on the optimal value function V^*. The lower bound $\underline{V}$ is represented by a collection of α-vector sets, $\{\Gamma_{\mathcal{Y}}(x) \mid x \in \mathcal{X}\}$, and initialized using fixed-action policies [3]. The upper bound $\overline{V}$ is represented by a collection of sets of belief-value pairs: $\{\Upsilon_{\mathcal{Y}}(x) \mid x \in \mathcal{X}\}$, where $\Upsilon_{\mathcal{Y}}(x) = \{(b_{\mathcal{Y}}, \overline{v}) \mid b_{\mathcal{Y}} \in \mathcal{B}_{\mathcal{Y}}(x)\}$. A belief-value pair $(b_{\mathcal{Y}}, \overline{v}) \in \Upsilon_{\mathcal{Y}}(x)$ gives an upper bound $\overline{v}$ on the value function at $(x, b_{\mathcal{Y}})$. They are used to perform sawtooth approximations [3] in each of the subspaces $\mathcal{B}_{\mathcal{Y}}(x)$. The upper bound can be initialized in various ways, using the MDP or the Fast Informed Bound technique [3].

The sampled points form a tree $T_{\mathcal{R}}$ (Fig. 2). Each node of $T_{\mathcal{R}}$ represents a sampled point in $\mathcal{R}$. In the following, we use the notation $(x, b_{\mathcal{Y}})$ to denote both a sampled point and its corresponding node in $T_{\mathcal{R}}$. The root of $T_{\mathcal{R}}$ is the initial belief point $(x_0, b_{\mathcal{Y}0})$.

To sample new belief points, we start from the root of $T_{\mathcal{R}}$ and traverse a single path down. At each node along the path, we choose a with the highest upper bound and choose x' and o that make the largest contribution to the gap ϵ between the upper and the lower bounds at the root of $T_{\mathcal{R}}$. New tree nodes are created if necessary. To do so, if at a node $(x, b_{\mathcal{Y}})$, we choose a, x', and o, a new belief on y is computed:

$$\begin{aligned} b'_{\mathcal{Y}}(y') &= \tau(x, b_{\mathcal{Y}}, a, x', o) \\ &= \eta Z(x', y', a, o) \\ &\quad \times \sum_{y \in \mathcal{Y}} T_{\mathcal{X}}(x, y, a, x') T_{\mathcal{Y}}(x, y, a, x', y') b_{\mathcal{Y}}(y), \end{aligned} \quad (6)$$

where η is a normalization constant. A new node $(x', b_{\mathcal{Y}}{}')$ is then inserted into $T_{\mathcal{R}}$ as a child of $(x, b_{\mathcal{Y}})$. Clearly, every point sampled this way is reachable from $(x_0, b_{\mathcal{Y}0})$. By carefully choosing a, x', and o based on the upper and lower bounds, we can keep the sampled belief points near $\mathcal{R}^*$.

When the sampling path ends under a suitable set of conditions, we go up back to the root of $T_{\mathcal{R}}$ along the same path and perform backup at each node along the way.

2) BACKUP*:* A backup operation at a node $(x, b_{\mathcal{Y}})$ collates the value function information in the children of $(x, b_{\mathcal{Y}})$ and propagates it back to $(x, b_{\mathcal{Y}})$. The operations are performed on both the lower and the upper bounds. For the lower bound, we perform α-vector backup (Algorithm 2). A new α-vector resulting from the backup operation at $(x, b_{\mathcal{Y}})$ is inserted into

Algorithm 2 α-vector backup at a node (x, b_y) of $T_{\mathcal{R}}$.

BACKUP$(T_{\mathcal{R}}, \Gamma, (x, b_y))$

1: For all $a \in \mathcal{A}$, $x' \in \mathcal{X}$, $o \in \mathcal{O}$,
$\alpha_{a,x',o} \leftarrow \text{argmax}_{\alpha \in \Gamma_y(x')}(\alpha \cdot \tau(x, b_y, a, x', o))$.
2: For all $a \in \mathcal{A}, y \in \mathcal{Y}$,
$\alpha_a(y) \leftarrow R(x, y, a) + \gamma \sum_{x',o,y'}(T_x(x, y, a, x')$
$\times T_y(x, y, a, x', y')Z(x', y', a, o)\alpha_{a,x',o}(y'))$.
3: $\alpha' \leftarrow \text{argmax}_{a \in \mathcal{A}}(\alpha_a \cdot b_y)$
4: Insert α' into $\Gamma_y(x)$.

$\Gamma_y(x)$, the set of α-vectors associated with observed state value x. For the upper bound backup at (x, b_y), we perform the standard Bellman update to get a new belief-value pair and insert it into $\Upsilon_y(x)$.

3) PRUNE*:* Invocation of SAMPLE and BACKUP generates new sampled points and α-vectors. However, not all of them are useful for constructing an optimal policy and are pruned to improve computational efficiency. We iterate through the α-vector sets in the collection $\{\Gamma_y(x) \mid x \in \mathcal{X}\}$ and prune any α-vector in $\Gamma_y(x)$ that does not dominate the rest at some sampled point (x, b_y), where $b_y \in \mathcal{B}_y(x)$.

Our description of the algorithm is quite brief due to space limitations. More details and justifications for our particular choices of the sampling, backup, and pruning strategies can be found in [9].

We would like to point out that to solve a MOMDP, the main modifications required in SARSOP are the belief update operation (Eq. (6)) and the backup operation (Algorithm 2). They are common to most point-based POMDP algorithms, such as PBVI [11], Perseus [20], HSVI [18], and FSVI [16]. So using Eq. (6) and Algorithm 2 to replace the corresponding parts in these algorithm would allow them to benefit from the MOMDP approach as well.

C. Correctness and Computational Efficiency

Modifying the belief update and backup operations does not affect the convergence property of SARSOP. The algorithm above provides the same theoretical guarantee as the original SARSOP algorithm. The formal statement and the proof are given in the full version of the paper.

MOMDPs allow the belief space $\mathcal{B}$ to be represented as a union of low-dimensional subspaces $\mathcal{B}_y(x)$ for $x \in \mathcal{X}$. This brings substantial computational advantages. Specifically, the efficiency gain of our algorithm comes mainly from BACKUP and PRUNE, where α-vectors are processed. In a MOMDP, α-vectors have length $|\mathcal{Y}|$, while in the corresponding POMDP, α-vectors have length $|\mathcal{X}||\mathcal{Y}|$. As a result, all operations on α-vectors in BACKUP and PRUNE are faster by a factor of $|\mathcal{X}|$ in our algorithm. Furthermore, in a MOMDP, α-vectors are divided into disjoint sets $\Gamma_y(x)$ for $x \in \mathcal{X}$. When we need to find the best α-vector, *e.g.*, in line 1 of Algorithm 2, we only do so within $\Gamma_y(x)$ for some fixed x rather than over all α-vectors, as in a POMDP.

The efficiency gain in BACKUP sometimes comes at a cost: although α-vectors in a MOMDP are shorter, they also contain less information, compared with POMDP α-vectors. In Algorithm 2, performing backup at (x, b_y) generates an α-vector that spans only the subspace $\mathcal{B}_y(x)$. The backup does not generate any information in any other subspace $\mathcal{B}_y(x')$ with $x' \neq x$. In contrast, POMDP algorithms do more computation while performing backup and generate α-vectors that span the entire space $\mathcal{B}$. If a problem has many similar observable states in the sense that the α-vectors in one belief subspace $\mathcal{B}_y$ are useful in other subspaces as well, then POMDP algorithms may obtain more useful information in each backup operation and perform better than our algorithm, despite the higher cost of each backup operation. This, however, requires a special property which may not hold in general for complex systems.

V. EXPERIMENTS

We used MOMDPs to model several distinct robotic tasks, all having large state spaces, and tested our algorithm on them. In this section, we describe the experimental setup and the results.

A. Robotic Tasks

1) Tag: The Tag problem first appeared in the work on PBVI [11], one of the first point-based POMDP algorithms. In Tag, the robot's goal is to follow a target that intentionally moves away. The robot and the target operate in an environment modeled as a grid. They can start in any grid positions, and in one step, they can either stay or move to one of four adjacent positions (above, below, left, and right). The robot knows its own position exactly, but can observe the target position only if they are in the same position. The robot pays a cost for each move and receives a reward when it arrives in the same position as that of the target.

In the MOMDP for this task, the robot position, which is known exactly, is modeled by the fully observable state variable x. The x variable can also take one extra value that indicates that the robot and the target are in the same position. The target position is modeled by the partially observable state variable y, as the robot does not see the target in general. Experiments were performed on environment maps with different resolutions. Tag(M) denotes an experiment on a map with M positions. Here $|\mathcal{X}| = M + 1$ and $|\mathcal{Y}| = M$, while in the standard POMDP model, the state space has $|\mathcal{S}| = (M + 1)M$ dimensions.

2) Rock Sample: The Rock Sample problem [18] has frequently been used to test the scalability of new POMDP algorithms. In this problem, a rover explores an area modeled as a grid and searches for rocks with scientific value. The rover always knows its own position exactly, as well as those of the rocks. However, it does not know which rocks are valuable. The rover can take noisy long-range sensor readings to gather information on the rocks. The accuracy of the readings depends on the distance between the rover and the rocks. The rover can also sample a rock in the immediate vicinity. It receives a reward or a penalty, depending on whether the sampled rock is valuable.

Here, the x variable in the MOMDP represents the robot position, and the y variable is a binary vector in which each entry indicates whether a rock is valuable or not. Experiments

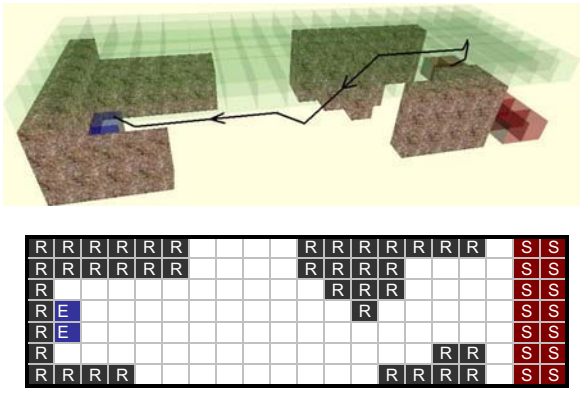

Fig. 3. AUV navigation. Top: The 3-D environment and an AUV simulation trajectory (marked in black) generated from the computed policy. The AUV can localize its horizontal position only at the surface level (shaded in light green). It rises to the surface level to localize, navigates through the rocks, and then dives to reach the goal. Bottom: The grid map for the deepest level. "S" marks the AUV starting positions, which are all located at this level. The AUV is equally likely to start in any of them. "E" marks the end positions, also located at this level only. "R" marks the rocks.

were performed on maps of different sizes and with different number of rocks. RockSample(M, R) denotes a map size of $M \times M$ and R rocks.

3) AUV Navigation: An AUV navigates in an oceanic environment modeled as a 3-D grid with 4 levels and 7×20 positions at each level (Fig. 3). It needs to navigate from the right boundary of the deepest level to some goal locations near the left boundary and must avoid rock formations, which are present in all levels except the surface. In each step, the AUV may either stay in the current position or move to any adjacent position along its current orientation. Whether the action is stay or move, the AUV may drift to a neighboring horizontal position due to control uncertainty or ocean currents. The AUV does not know its exact starting position. It knows its horizontal position only at the surface level, where GPS signals are available. However, surfacing causes heavy fuel consumption and must be avoided if possible. Using its pressure sensor and gyroscope, the AUV can acquire accurate information on the depth and the orientation, which is discretized into 24 values.

In the MOMDP model, the x variable represents the AUV's depth and orientation, and the y variable represents the AUV's horizontal position. Although in our specific setup, the belief over the AUV's horizontal position is always unimodal, the MOMDP approach is general and can be applied without change if the belief is multimodal.

B. Results

We applied the MOMDP algorithm to the three tasks above. For each task, 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 algorithm for a maximum of half an hour to reach this level. To estimate the expected total reward of the resulting policy, we performed sufficiently large number of simulation runs until the variance in the estimated value was small. For comparison, we also ran SARSOP [9], a leading POMDP algorithm, on the same tasks modeled as standard POMDPs. Both algorithms are implemented in C++, and the experiments were performed on a PC with a 2.66GHz Intel processor and 2GB memory.

TABLE I

PERFORMANCE COMPARISON ON TASKS WITH MIXED OBSERVABILITY.

		Reward	Time (s)
Tag(29)			
$\lvert\mathcal{X}\rvert$=30,$\lvert\mathcal{Y}\rvert$=29	MOMDP	-6.03 ± 0.04	4.7
$\lvert\mathcal{S}\rvert$=870, $\lvert\mathcal{A}\rvert$=5, $\lvert\mathcal{O}\rvert$=30	SARSOP	-6.03 ± 0.12	16.5
Tag(55)			
$\lvert\mathcal{X}\rvert$=56,$\lvert\mathcal{Y}\rvert$=55	MOMDP	-9.90 ± 0.11	19
$\lvert\mathcal{S}\rvert$=3,080, $\lvert\mathcal{A}\rvert$=5, $\lvert\mathcal{O}\rvert$=56	SARSOP	-9.90 ± 0.12	736
RockSample(7,8)			
$\lvert\mathcal{X}\rvert$=50,$\lvert\mathcal{Y}\rvert$=256	MOMDP	21.47 ± 0.04	160
$\lvert\mathcal{S}\rvert$=12,545‡, $\lvert\mathcal{A}\rvert$=13, $\lvert\mathcal{O}\rvert$=2	SARSOP	21.39 ± 0.01	810
RockSample(10,10)			
$\lvert\mathcal{X}\rvert$=101,$\lvert\mathcal{Y}\rvert$=1,024	MOMDP	21.47 ± 0.04	318
$\lvert\mathcal{S}\rvert$=102,401, $\lvert\mathcal{A}\rvert$=15, $\lvert\mathcal{O}\rvert$=2	SARSOP	21.47 ± 0.11	1589
RockSample(11,11)			
$\lvert\mathcal{X}\rvert$=122,$\lvert\mathcal{Y}\rvert$=2,048	MOMDP	21.80 ± 0.04	188
$\lvert\mathcal{S}\rvert$=247,809, $\lvert\mathcal{A}\rvert$=16, $\lvert\mathcal{O}\rvert$=2	SARSOP	21.56 ± 0.11	1369
AUV Navigation			
$\lvert\mathcal{X}\rvert$=96,$\lvert\mathcal{Y}\rvert$=141	MOMDP	1020.0 ± 8.5	124
$\lvert\mathcal{S}\rvert$=13,536, $\lvert\mathcal{A}\rvert$=6, $\lvert\mathcal{O}\rvert$=144	SARSOP	1019.8 ± 9.7	409

1) Mixed Observability Tasks: The results are shown in Table I. Column 3 of the table lists the estimated expected total rewards for the computed policies and the 95% confidence intervals. Column 4 lists the running times.

For all tasks, the MOMDP algorithm obtained good approximate solutions well within the time limit and outperformed SARSOP by many times. We ran multiple experiments for Rock Sample with different map sizes and numbers of rocks. As the problem size increases, both $|\mathcal{X}|$ and $|\mathcal{Y}|$ increase. The performance gap between our algorithm, which uses the MOMDP model, and SARSOP, which uses the standard POMDP model, tends to increase as well. In particular, for the largest problem, RockSample(11,11), SARSOP never reached the same reward level attained by the MOMDP algorithm within the time limit. This is not surprising as the computational efficiency gain achievable from the MOMDP model depends on $|\mathcal{X}|$ (Section IV-C). The experiments on Tag show a similar trend.

2) Pseudo Full Observability: We also tested our algorithm on robotic tasks in which all sensed state components are partially observable and obtained promising results.

The original Tag problem is modified to create a noisy version, in which the robot has p% chance of observing its own position correctly and $(1/8)(100 - p)$% chance of observing each of the 8 surrounding positions. In this case, both the robot position and the target position are partially observable. However, applying the technique in Section III-B.1, we modeled the robot position by the fully observable state variable x in the MOMDP and solved the MOMDP with our algorithm. The value function computed by the algorithm was used to generate a policy to control the robot without assuming any fully observable state variables.

We ran experiments with different sensor accuracy p for the Noisy Tag problem. The results are shown in Table II.

‡Small optimizations can be performed to reduce the number of states in the POMDP model. So $|\mathcal{S}|$ may not be equal to $|\mathcal{X}||\mathcal{Y}|$ for the corresponding MOMDP. However, this is not significant enough to affect $|\mathcal{S}|{=}O(|\mathcal{X}||\mathcal{Y}|)$.

TABLE II

PERFORMANCE COMPARISON ON TASKS USING PSEUDO OR REPARAMETERIZED FULL OBSERVABILITY TECHNIQUES.

		Reward	Time (s)
NoisyTag(29,90%)			
$\|\mathcal{X}\|$=30,$\|\mathcal{Y}\|$=29	MOMDP	-11.12 ± 0.14	4.5
$\|\mathcal{S}\|$=870, $\|\mathcal{A}\|$=5, $\|\mathcal{O}\|$=30	SARSOP	-11.12 ± 0.14	228.0
NoisyTag(29,50%)			
$\|\mathcal{X}\|$=30,$\|\mathcal{Y}\|$=29	MOMDP	-12.14 ± 0.14	1.5
$\|\mathcal{S}\|$=870, $\|\mathcal{A}\|$=5, $\|\mathcal{O}\|$=30	SARSOP	-12.15 ± 0.14	11.6
NoisyTag(29,10%)			
$\|\mathcal{X}\|$=30,$\|\mathcal{Y}\|$=29	MOMDP	-12.53 ± 0.14	1.5
$\|\mathcal{S}\|$=870, $\|\mathcal{A}\|$=5, $\|\mathcal{O}\|$=30	SARSOP	-12.59 ± 0.14	176.4
NoisyTag(55, 3×3)			
$\|\mathcal{X}\|$=56,$\|\mathcal{Y}\|$=495	MOMDP	-10.62 ± 0.10	32
$\|\mathcal{S}\|$=3,080, $\|\mathcal{A}\|$=5, $\|\mathcal{O}\|$=2	SARSOP	-10.61 ± 0.08	927

NoisyTag$(M, p\%)$ denotes a modified Tag problem with a map of M positions and sensor accuracy $p\%$. The MOMDP algorithm drastically outperformed SARSOP, even when p was as low as 10%. This is a little surprising. The MOMDP model brings computational advantages, but is less accurate than the POMDP model, due to the assumption of fully observable state variables. One would expect that the MOMDP algorithm may reach a reasonable reward level faster, but lose to SARSOP in the long run. However, in this case we did not observe any significant performance loss for the MOMDP algorithm. To confirm the results, we ran SARSOP for two additional hours, but its reward level did not improve much beyond those reported in the table.

3) Reparameterized Full Observability: The Tag problem is again modified so that the robot never observes its own position exactly, but only the 3×3 region around it in the grid. Both the robot and the target positions are partially observable. However, the preimage of any observation on the robot position is always bounded. We can apply the approach described in Section III-B.2 and reparameterize the robot position as $(o_{\mathrm{r}}, \delta_{\mathrm{r}})$, where o_{r} indicates a 3×3 region in the grid and δ_{r} indicates the actual position of the robot within the region. We then model the reparameterized problem as a MOMDP: the x variable represents o_{r}, and the y variable represents δ_{r} and the target position.

Again the MOMDP algorithm significantly outformed SARSOP (Table II). This suggests that extracting fully observable state variables through reparameterization is a promising idea and deserves further investigation.

VI. CONCLUSION

POMDPs have been successfully used for motion planning under uncertainty in various robotic tasks [5], [11], [12], [19]. A major challenge remaining is to scale up POMDP algorithms for complex robotic systems. Exploiting the fact that many robotic systems have mixed observability, our MOMDP approach uses a factored model to separate the fully and partially observable components of a robot's state. We show that the factored representation drastically improves the speed of POMDP planning, when combined with a point-based POMDP algorithm. We further show that even when a robot does not have obvious fully observable state components, it still can be modeled as a MOMDP by reparameterizing the robot's state space.

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 Yanzhu Du for helping with the software implementation. We also thank Tomás Lozano-Pérez and Leslie Kaelbling from MIT for many insightful discussions. This work is supported in part by AcRF grant R-252-000-327-112 from the Ministry of Education of Singapore.

REFERENCES

[1] C. Guestrin, D. Koller, and R. Parr. Solving factored POMDPs with linear value functions. In *Int. Jnt. Conf. on Artificial Intelligence Workshop on Planning under Uncertainty & Incomplete Information*, pp. 67–75, 2001.

[2] C. Guestrin, D. Koller, R. Parr, and S. Venkataraman. Efficient solution algorithms for factored MDPs. *J. Artificial Intelligence Research*, 19:399–468, 2003.

[3] M. Hauskrecht. Value-function approximations for partially observable Markov decision processes. *J. Artificial Intelligence Research*, 13:33–94, 2000.

[4] M. Hauskrecht and H. Fraser. Planning medical therapy using partially observable Markov decision processes. In *Proc. Int. Workshop on Principles of Diagnosis*, pp. 182–189, 1998.

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

[6] D. Hsu, W. Lee, and N. Rong. What makes some POMDP problems easy to approximate? In *Advances in Neural Information Processing Systems (NIPS)*, 2007.

[7] ———. A point-based POMDP planner for target tracking. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pp. 2644–2650, 2008.

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

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

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

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

[12] J. Pineau, M. Montemerlo, M. Pollack, N. Roy, and S. Thrun. Towards robotic assistants in nursing homes: Challenges and results. *Robotics & Autonomous Systems*, 42(3–4):271–281, 2003.

[13] P. Poupart and C. Boutilier. Value-directed compression of POMDPs. In *Advances in Neural Information Processing Systems (NIPS)*, 15:1547–1554. The MIT Press, 2003.

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

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

[16] G. Shani, R. Brafman, and S. Shimony. Forward search value iteration for POMDPs. In *Proc. Int. Jnt. Conf. on Artificial Intelligence*, 2007.

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

[18] T.Smith and R.Simmons. Heuristic search value iteration for POMDPs. In *Proc. Uncertainty in Artificial Intelligence*, pp. 520–527, 2004.

[19] ———. Point-based POMDP algorithms: Improved analysis and implementation. In *Proc. Uncertainty in Artificial Intelligence*, 2005.

[20] M. Spaan and N. Vlassis. A point-based POMDP algorithm for robot planning. In *Proc. IEEE Int. Conf. on Robotics & Automation*, 2004.