TO APPEAR IN *International Jounral of Robotics Research*

Planning under Uncertainty for Robotic Tasks with Mixed Observability

Sylvie C.W. Ong
Department of Computer Science, National University of Singapore
Singapore 117417, Singapore
song@comp.nus.edu.sg

Shao Wei Png
School of Computer Science, McGill University
Montreal, Quebec H3A 2A7, Canada
shaowei.png@mail.mcgill.ca

David Hsu
Department of Computer Science, National University of Singapore
Singapore 117417, Singapore
dyhsu@comp.nus.edu.sg

Wee Sun Lee
Department of Computer Science, National University of Singapore
Singapore 117417, Singapore
leews@comp.nus.edu.sg

Abstract

Partially observable Markov decision processes (POMDPs) provide a principled, general framework for robot motion planning in uncertain and dynamic environments. They have been applied to various robotic tasks. However, solving POMDPs exactly is computationally intractable. A major challenge is to scale up POMDP algorithms for complex robotic tasks.

Robotic systems often have *mixed observability*: even when a robot's state is not fully observable, some components of the state may still be so. 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. This factored representation can be combined with any point-based algorithm to compute approximate POMDP solutions. Experimental results show that on standard test problems, our approach improves the performance of a leading point-based POMDP algorithm by many times.

1 Introduction

Motion planning under uncertainty is a critical ability for autonomous robots operating in uncontrolled environments, such as homes or offices. For robotic systems, uncertainty arises from two main sources: robot control and sensing. If a robot's control is imperfect, but its state is fully observable due to accurate sensing, 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.*, [5]). When a robot's state is not fully observable, possibly due to noisy sensors, partially observable Markov decision processes (POMDPs) become necessary, but solving POMDPs exactly is computationally intractable [15]. Despite the impressive progress of point-based POMDP algorithms in recent years [14, 16, 22, 26, 27], 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 so. For example, consider a mobile robot equipped with a compass, but not a geographic positioning system (GPS). Its orientation is fully observable, though its position may only be partially observable. We call such problems *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 resulting MOMDP model leads to much faster planning algorithms 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 a robot's *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, one main contributor to the difficulty of solving POMDPs is the "curse of dimensionality": in a discrete-state POMDP, $\mathcal{B}$ 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 has 100 possible positions on a 10×10 grid in the horizontal plane, 5 depth levels, and 25 orientations. The resulting belief space is 12,500-dimensional!

However, if the robot has an accurate pressure sensor and a gyroscope, we may assume that the depth level and the orientation are fully observable and maintain a belief on the robot's uncertain horizontal position only. The space $\mathcal{B}$ then becomes a union of 125 disjoint 100-dimensional subspaces. Each subspace represents an exact depth level, an exact orientation, and beliefs on the uncertain horizontal positions. These 100-dimensional subspaces are still large, but a significant reduction from the original 12,500-dimensional space.

To exploit full observability for computational efficiency, we thus separate the fully and partially observable state components using a factored model and represent the disjoint belief subspaces in a MOMDP explicitly 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. We consider two common types of sensor limitations here. In the first case, some state components are sensed accurately and can be considered fully observable while other state components are not. We then separate the state components with high sensing accuracy from the rest in the MOMDP representation. 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 this case can be modeled as a MOMDP by (re)parameterizing the state space. The reparameterization technique enables a much broader class of planning problems to benefit from MOMDP modeling.

We tested our new approach on a set of distinct robotic tasks with large state spaces. The results show that it improves the performance of a leading point-based POMDP algorithm by many times.

In the following, we start with some background on POMDPs (Section 2). We then define the MOMDP model and illustrate how it can be used (Section 3). In Section 4, we combine the MOMDP representation with a point-based POMDP algorithm and analyze the benefits. We report the simulation experiments in Section 5. Finally, we summarize the main results in Section 7.

2 Background

2.1 POMDPs

A POMDP models an agent taking a sequence of actions under uncertainty to maximize its total reward. Formally a discrete-state, infinite-horizon, discounted POMDP 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 takes an action $a \in \mathcal{A}$ and moves from a state $s \in \mathcal{S}$ to $s' \in \mathcal{S}$. 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 $o \in \mathcal{O}$ is again described as a conditional probability function $Z(s', a, o) = p(o|s', a)$ for $s' \in \mathcal{S}$ and $a \in \mathcal{A}$. See Figure 1 for an illustration.

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 $s \in \mathcal{S}$ and takes action $a \in \mathcal{A}$. The agent's goal 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 given by $\mathrm{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$, where s_t and a_t denote the agent's state and action at time t, respectively.

For POMDPs, 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 a belief. A POMDP policy $\pi \colon \mathcal{B} \to \mathcal{A}$ maps a belief $b \in \mathcal{B}$, which is a probability distribution over $\mathcal{S}$, to the prescribed action $a \in \mathcal{A}$.

A policy π induces a value function V, which specifies the expected total reward $V(b)$ 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 vector representing a belief and Γ is a finite set of vectors called *α-vectors* [24]. 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, using (1). So a policy can be represented as 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 repeatedly executes two steps. The first step is action selection. 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 update. After taking an action a and receiving an observation o, the agent updates its belief:

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

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

Eq. (2) shows that a POMDP can be viewed as a belief-space MDP. Each state of this MDP represents a belief, resulting in a continuous state space. The transition function can be easily constructed using (2).

More information about POMDPs is available in [13], [28].

2.2 Related Work

POMDPs are a powerful framework for planning under uncertainty [13, 24]. It has solid mathematical foundations and wide applicability. The main disadvantage is high computational complexity [15]. As mentioned in Section 1, the belief space $\mathcal{B}$ used in 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 POMDP planning. There are several approaches to overcome this difficulty, including sampling $\mathcal{B}$, building lower-dimensional approximations of $\mathcal{B}$ [19], and employing structured value function representations [4, 6, 18].

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 [14, 16, 22, 26, 27]. They have been successfully applied to several moderately complex robotic tasks, including navigation [2, 20], grasping [10], target tracking [12, 16], and exploration [26]. In some cases, POMDPs with hundreds of states have been solved in a matter of seconds (see, *e.g.*, [12, 26]). POMDPs have also been applied to many domains beyond robotics, *e.g.*, assistive technology [9] and dialog management [23, 29].

A popular structured value function representation is the algebraic decision diagram (ADD) [6]. It exploits context-specific independence to aggregate states with the same expected total rewards and achieves a more efficient value function representation. While ADDs are effective for some application domains, its suitability for robotic tasks is unclear. In a typical robotic task, all robot actions incur costs. Since the expected total reward is discounted, it is unlikely to have many states with the same or even similar values. Effective aggregation then becomes difficult.

Our work on MOMDPs aims at alleviating the difficulty of high-dimensional belief spaces and scaling up point-based algorithms for realistic robotic tasks. The main idea is that a MOMDP model factors out the fully and partially observable components of an agent's state and leads to a compact lower-dimensional representation of $\mathcal{B}$. We then combine this new representation with a point-based algorithm and obtain a much faster planning algorithm. A related idea has been developed earlier for medical therapy planning [8]. Although MOMDPs take advantage of factorization just as ADDs do, they exploit different underlying problem structure. To represent α-vectors compactly, ADDs rely on "homogeneity" to aggregate states with the same or similar expected total rewards. In contrast, MOMDPs exploit the observability of state variables.

Many of the above ideas are complementary. For example, we combine a point-based algorithm with the MOMDP model. Point-based algorithms can also benefit from the ADD representation [22]. For a suitable application, it may even be beneficial to use ADDs to represent α-vectors along with a MOMDP model.

3 Mixed Observability MDPs

3.1 MOMDP models

In a standard POMDP model, the state lumps together multiple components. Consider again our AUV navigation example from Section 1. A single state variable models the robot's horizontal position, depth, and orientation. In contrast, a factored POMDP model separates the multiple state components and represents each

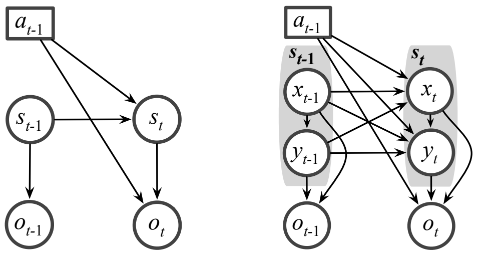

Figure 1: The POMDP model (left) and the MOMDP model (right). A MOMDP state s is factored into two variables: $s = (x, y)$, where x is fully observable and y is partially observable.

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 the 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.

We propose to represent a robotic system with mixed observability as a factored POMDP with mixed state variables and call our 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 values for x and $\mathcal{Y}$ is the space of all values for y. In our AUV example, $x = (d, \theta)$ represents the depth and the orientation, and $y = p$ represents the horizontal position.

Formally a MOMDP model is specified as a tuple $(\mathcal{X}, \mathcal{Y}, \mathcal{A}, \mathcal{O}, T_\mathcal{X}, T_\mathcal{Y}, Z, R, \gamma)$. The transition function

$$T_\mathcal{X}(x, y, a, x') = p(x'|x, y, a)$$

gives the probability that the fully observable state variable has value x' if the robot takes action a in state (x, y), and

$$T_\mathcal{Y}(x, y, a, x', y') = p(y'|x, y, a, x')$$

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 resulting value x'. Compared with the POMDP model, the MOMDP model has a factored state space $\mathcal{X} \times \mathcal{Y}$, with transition functions $T_\mathcal{X}$ and $T_\mathcal{Y}$, while other aspects remain the same. See Figure 1 for a comparison. As mentioned in Section 2.1, a POMDP can be viewed as an MDP with continuous (belief) states. Correspondingly, a MOMDP can be viewed as a hybrid MDP with both discrete and continuous states [1, 3]. The discrete state variable corresponds to x, and the continuous state variable corresponds to $b_\mathcal{Y}$, the belief over y.

So far, the changes introduced by the MOMDP model seem mostly notational. The computational advantages become clear when we consider the belief space $\mathcal{B}$. Since x is fully observable and known exactly, we only need to maintain a belief $b_\mathcal{Y}$, a probability distribution on y. Any belief $b \in \mathcal{B}$ on the complete system state $s = (x, y)$ is then represented as $(x, b_\mathcal{Y})$.

Consider a simple example, in which an agent's state is specified by two binary variables (x, y). The full belief space is a 3-simplex, *i.e.*, a tetrahedron, because the probabilities of all the states must sum up to 1. Each corner of the tetrahedron represents a belief that the agent is in any of the four states $(0, 0), (0, 1), (1, 0),$ and $(1, 1)$ with probability 1. The center of the tetrahedron represents the belief that the system is in the four states with equal probabilities. Now if x is fully observable, then all beliefs must lie on one of the two opposite edges of the tetrahedron. So $\mathcal{B}$ is in fact a union of two line segments and not the full tetrahedron (Figure 2).

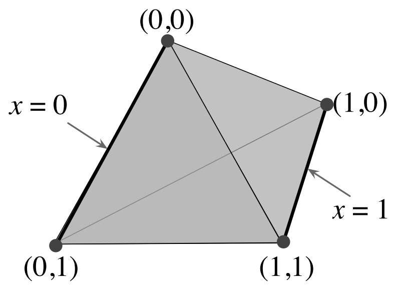

Figure 2: The belief space for two binary variables (x, y) is a tetrahedron. If x is fully observable, then all beliefs lie on two edges of the tetrahedron, corresponding to $x = 0$ and $x = 1$.

In general, let $\mathcal{B}_{\mathcal{Y}}$ denote the space of all beliefs on the partially observable state variable y. We associate with each value x of the fully observable state variable a belief space for y: $\mathcal{B}_{\mathcal{Y}}(x) = \{(x, b_{\mathcal{Y}}) \mid b_{\mathcal{Y}} \in \mathcal{B}_{\mathcal{Y}}\}$. $\mathcal{B}_{\mathcal{Y}}(x)$ is a subspace in $\mathcal{B}$, and $\mathcal{B}$ is a union of these subspaces: $\mathcal{B} = \bigcup_{x \in \mathcal{X}} \mathcal{B}_{\mathcal{Y}}(x)$. While $\mathcal{B}$ has $|\mathcal{X}| \times |\mathcal{Y}|$ dimensions, where $|\mathcal{X}|$ and $|\mathcal{Y}|$ are the number of states in $\mathcal{X}$ and $\mathcal{Y}$, each $\mathcal{B}_{\mathcal{Y}}(x)$ has only $|\mathcal{Y}|$ dimensions. Effectively we represent the high-dimensional space $\mathcal{B}$ as a union of lower-dimensional subspaces. When the uncertainty of 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 belief space representation and the resulting implications for policy representation and computation.

3.2 MOMDP Policies

Now let us consider how we would represent and execute a MOMDP policy. As mentioned in Section 2.1, 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 $(x, b_{\mathcal{Y}})$, and $\mathcal{B}$ is represented as a union of subspaces $\mathcal{B}_{\mathcal{Y}}(x)$ for $x \in \mathcal{X}$. Correspondingly, a MOMDP value function $V(x, b_{\mathcal{Y}})$ is represented as a collection of α-vector sets: $\{\Gamma_{\mathcal{Y}}(x) \mid x \in \mathcal{X}\}$, where for each x, $\Gamma_{\mathcal{Y}}(x)$ is a set of α-vectors defined over $\mathcal{B}_{\mathcal{Y}}(x)$. To evaluate $V(x, b_{\mathcal{Y}})$, we first use the value of x as an index to find the right α-vector set and then find the maximum α-vector from the set:

$$V(x, b_{\mathcal{Y}}) = \max_{\alpha \in \Gamma_{\mathcal{Y}}(x)} \{\alpha \cdot b_{\mathcal{Y}}\}. \tag{3}$$

It is not difficult to see that any value function $V(b) = \max_{\alpha \in \Gamma}(\alpha \cdot b)$ can be represented in this new form. Geometrically, each α-vector set $\Gamma_{\mathcal{Y}}(x)$ represents a *restriction* of $V(b)$ to the subspace $\mathcal{B}_{\mathcal{Y}}(x)$, obtained by restricting the domain of $V(b)$ from $\mathcal{B}$ to $\mathcal{B}_{\mathcal{Y}}(x)$. In MOMDP policy computation, we compute only these restrictions in lower-dimensional subspaces $\mathcal{B}_{\mathcal{Y}}(x)$ for $x \in \mathcal{X}$, because $\mathcal{B}$ is simply a union of these subspaces.

A comparison of (1) and (3) indicates that (3) results in faster policy execution, because action selection can be performed more efficiently. 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 $\Gamma_{\mathcal{Y}}(x)$, which is potentially much smaller than Γ.

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 model uncertainty commonly encountered in robotic systems.

3.3 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, often some components are fully observable, due to accurate sensing, while others are not. This is a natural case for modeling with MOMDPs. All fully observable state components are grouped together and modeled by the variable x. The other components are modeled by the variable y.

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. We describe two techniques below.

3.3.1 Pseudo Full Observability

All sensors are ultimately limited in resolution. It is task-dependent to determine whether the sensor resolution is accurate enough to make a 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, because 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 exploit the MOMDP model for faster policy computation and execution. However, treating the unseen target's position as fully observable is not reasonable and unlikely to lead to a useful policy.

To improve the robustness of robot control, we can actually execute a computed MOMDP policy on the original POMDP model, which makes no assumption of fully observable state variables. The POMDP model treats both variables x and y as partially observable and maintains a belief b over (x, y). To account for the additional uncertainty on x, our idea is to define a new value function $V'(b)$ by averaging over the computed MOMDP value function $V(x, b_{\mathcal{Y}})$ and use $V'(b)$ for policy execution rather than use the MOMDP value function directly. 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

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

can be used for policy execution under the POMDP model 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 $V'(b)$ constructed from $V^*(x, b_{\mathcal{Y}})$ is an upper bound on $V^*(b)$. In this usage, the MOMDP model becomes an approximation to the POMDP model. It is less accurate due to the additional assumption, but has substantial computational advantages.

3.3.2 Reparameterized Full Observability

Instead of assuming full observability, a more subtle way of exploiting MOMDPs is to reparameterize the state space $\mathcal{S}$.

We first illustrate the reparameterization technique on a robot navigation task, modeled as a standard POMDP $(\mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R, \gamma)$. Suppose that the robot's position sensor is noisy and localizes the robot to a small region around the actual position in the plane. The state $s \in \mathcal{S}$ specifies the robot position and $o \in \mathcal{O}$

is the observed robot position. If we parameterize $\mathcal{S}$ with the standard Cartesian coordinates, none of the coordinates would be fully observable. Instead, we reparameterize $\mathcal{S}$ so that $s = (x, y)$, where $x = o$, the observed robot position, and y is the offset of the actual position from the observed position o. Now x is clearly fully observable by definition. Using this parameterization, we can construct the new transition functions $T_{\mathcal{X}}$ and $T_{\mathcal{Y}}$ from the old transition function T and observation function Z:

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

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

where $s = (x, y)$ and $s' = (x', y')$. The correctness of this construction can be verified by applying the basic rules of probability as well as the definitions of T, $T_{\mathcal{X}}$, and $T_{\mathcal{Y}}$. We also construct the new observation function

$$Z_{\text{M}}(x', y', a, o) = \begin{cases} 1 & \text{if } x' = o \\ 0 & \text{if } x' \neq o \end{cases} \tag{6}$$

and the new reward function

$$R_{\text{M}}(x, y, a) = R(s, a) \tag{7}$$

where $s = (x, y)$.

To reparameterize $\mathcal{S}$ in the general case, we again set $s = (x, y)$ and $x = o$. To determine y, consider the *preimage* $h(o)$ of an observation o. We define $h(o)$ as the set of states that have non-zero probability of emitting o. If o is in fact a combination of observations from multiple sensors, each with its own preimage, then $h(o)$ is the intersection of the individual preimages. The state variable y specifies the exact state within the preimage $h(x)$. Now the transition, observation, and reward functions of the reparameterized MOMDP can be constructed according to (4–7).

Theorem 1 below states formally the equivalence between the POMDP and the reparameterized MOMDP. It says that the belief space dynamics is exactly the same under both models, and so is the expected total reward for any policy. The proof is given in the appendix.

Theorem 1 *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:*

(1) Given the same initial belief and any sequence of actions and observations, the beliefs reached under both models are the same.

(2) Given the same initial belief, any policy has the same expected total reward under both models.

Compared with pseudo full observability, reparameterized full observability results in an MOMDP equivalent to the original POMDP and thus does not sacrifice solution quality in any way. The reparameterization can be performed on any POMDP, but when does the resulting MOMDP bring computational advantages? To understand this, define 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 $c < 1$. When a system has bounded uncertainty with a small constant c, then $|\mathcal{Y}|$ is small for the reparameterized MOMDP, and this leads to significant improvement in computational efficiency. To determine whether reparameterization is beneficial, we can bound the constant c by calculating the maximum size of preimages.

Algorithm 1 Point-based MOMDP policy computation.

```
1: Initialize the $\alpha$-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 belief-value pairs, $\Upsilon = \{\Upsilon_\mathcal{Y}(x) \mid x \in \mathcal{X}\}$, representing 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($T_\mathcal{R}$, $\Gamma$, $\Upsilon$).
5:     Choose a subset of nodes from $T_\mathcal{R}$. For each chosen node $(x, b_\mathcal{Y})$, BACKUP-LB$(T_\mathcal{R}, \Gamma, (x, b_\mathcal{Y}))$ and BACKUP-UB$(T_\mathcal{R}, \Upsilon, (x, b_\mathcal{Y}))$.
6:     PRUNE($T_\mathcal{R}$, $\Gamma$).
7: until termination conditions are satisfied.
8: return $\Gamma$.
```

4 Computing MOMDP Policies

4.1 Overview

MOMDPs enable us to represent a high-dimensional belief space $\mathcal{B}$ as a union of lower-dimensional subspaces and restrict value function computation to these subspaces rather than the entire $\mathcal{B}$. We can combine this representation with most of the existing POMDP algorithms that use α-vectors for value function and policy representation, and improve their performance. However, for concreteness, we present our approach based on the SARSOP algorithm [14], one of the fastest point-based POMDP algorithms.

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 them as an approximate representation of $\mathcal{B}$, rather than represent $\mathcal{B}$ exactly. They also maintain a set of α-vectors as an approximation to the optimal value function V^*.

Specifically the SARSOP algorithm makes use of *value iteration* [21]. 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.

Our point-based MOMDP algorithm samples incrementally a set of points from $\mathcal{B}$ and maintains a collection of α-vector sets, which represent a piecewise-linear lower-bound approximation $\underline{V}$ to V^*. It also maintains an upper bound $\overline{V}$ on V^*. To improve the bounds, the algorithm performs backup operations on $\underline{V}$ and $\overline{V}$ at the sampled points. A backup operation is essentially an iteration of dynamic programming, which improves the approximation by looking ahead one step further. Under suitable conditions, $\underline{V}$ converges to V^* [11, 16, 26].

4.2 Algorithm

We give only a brief description of the overall algorithm here, with a focus on illustrating the use of the MOMDP representation (Algorithm 1).

The algorithm starts by initializing $\underline{V}$ and $\overline{V}$. The lower bound $\underline{V}$ is represented as a collection of α-vector sets, $\{\Gamma_\mathcal{Y}(x) \mid x \in \mathcal{X}\}$, and initialized using fixed-action policies [7]. 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 $(x, b_\mathcal{Y})$. For each subspace $\mathcal{B}_\mathcal{Y}(x)$, we interpolate the belief-value pairs in $\Upsilon_\mathcal{Y}(x)$ through sawtooth approximation [7] and produce an upper bound

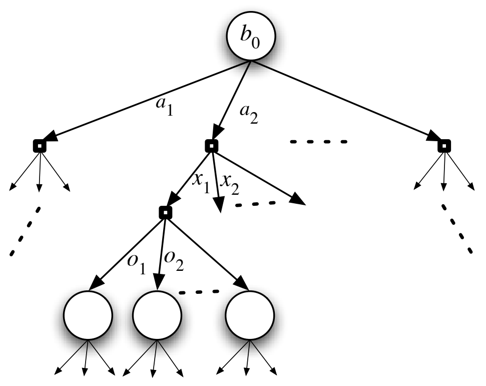

Figure 3: The belief search tree rooted at $b_0 = (x_0, b_{\mathcal{Y}0})$. Each node represents a MOMDP belief state $(x, b_{\mathcal{Y}})$.

on $V^*(x, b_{\mathcal{Y}})$ over $\mathcal{B}_{\mathcal{Y}}(x)$. The upper bound can be initialized in various ways, using, for example, the MDP or the Fast Informed Bound (FIB) technique [7]. After initialization, our algorithm iterates over three main functions, SAMPLE, BACKUP, and PRUNE.

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 and observations. 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 sequences of *optimal* actions, usually a much smaller space than $\mathcal{R}$. Since $\mathcal{R}^*$ is unknown in advance, we compute successive approximations of $\mathcal{R}^*$ and converge to it iteratively.

The sampled points form a tree $T_{\mathcal{R}}$ (Figure 3). 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 action a with the highest upper bound and choose state x' and observation o that make the largest contribution to the gap ϵ between the upper and lower bounds at the root of $T_{\mathcal{R}}$. The choice of a is optimistic with respect to the upper bound of V^*, similar to that in the A* algorithm [21]. The choices of x and o are a heuristic to help the algorithm converge faster. New tree nodes are created, if necessary, at the bottom of $T_{\mathcal{R}}$. To do so, if at a node $(x, b_{\mathcal{Y}})$, we choose a, o and x' and compute a new belief $b'_{\mathcal{Y}}$ on y:

$$\begin{aligned} b'_{\mathcal{Y}}(y') &= \tau_{\mathrm{M}}(x, b_{\mathcal{Y}}, a, o, x') \\ &= \eta Z(x', y', a, o) \sum_{y \in \mathcal{Y}} T_{\mathcal{X}}(x, y, a, x') T_{\mathcal{Y}}(x, y, a, x', y') b_{\mathcal{Y}}(y), \end{aligned} \tag{8}$$

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 belief point sampled this way is reachable from $(x_0, b_{\mathcal{Y}0})$.

The sampling path terminates under a suitable set of conditions. These conditions make use of the upper and lower bounds as well as a simple on-line learning technique to estimate whether points further down the path reduce ϵ, thus likely lying close to $\mathcal{R}^*$ and worth further exploration. When a sampling path terminates, we go back up this path to the root of $T_{\mathcal{R}}$ and perform backup operations at each node along the way.

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 bound $\underline{V}$ and the upper bound $\overline{V}$. For $\underline{V}$, we perform α-vector backup (Algorithm 2). A new α-vector resulting from the backup at $(x, b_{\mathcal{Y}})$ is inserted into $\Gamma_{\mathcal{Y}}(x)$, the set of α-vectors associated with observed state value x. For the upper bound backup at $(x, b_{\mathcal{Y}})$, we perform the standard Bellman update to get a new belief-value pair $(b_{\mathcal{Y}}, \overline{v})$ and insert it into $\Upsilon_{\mathcal{Y}}(x)$, the set of belief-value pairs associated with observed state value x (Algorithm 3).

Algorithm 2 Lower bound backup at a node $(x, b_{\mathcal{Y}})$ of $T_{\mathcal{R}}$.

```
BACKUP-LB($T_{\mathcal{R}}, \Gamma, (x, b_{\mathcal{Y}})$)
1: For all $a \in \mathcal{A}$, $o \in \mathcal{O}$, and $x' \in \mathcal{X}$, $\alpha_{a,o,x'} \leftarrow \text{argmax}_{\alpha \in \Gamma_{\mathcal{Y}}(x')}(\alpha \cdot \tau_{\text{M}}(x, b_{\mathcal{Y}}, a, o, x'))$.
2: For all $y \in \mathcal{Y}$ and $a \in \mathcal{A}$, $\alpha_a(y) \leftarrow R(x, y, a) + \gamma \sum_{o,x',y'} (T_{\mathcal{X}}(x, y, a, x')$
       $\times T_{\mathcal{Y}}(x, y, a, x', y') Z(x', y', a, o) \alpha_{a,o,x'}(y'))$.
3: $\alpha' \leftarrow \text{argmax}_{a \in \mathcal{A}}(\alpha_a \cdot b_{\mathcal{Y}})$.
4: Insert $\alpha'$ into $\Gamma_{\mathcal{Y}}(x)$.
```

Algorithm 3 Upper bound backup at a node $(x, b_{\mathcal{Y}})$ of $T_{\mathcal{R}}$.

```
BACKUP-UB($T_{\mathcal{R}}, \Upsilon, (x, b_{\mathcal{Y}})$)
1: For all $a \in \mathcal{A}, o \in \mathcal{O}$, and $x' \in \mathcal{X}$, compute the upper bound $\overline{v}_{a,o,x'}$ at the belief $b'_{\mathcal{Y}} = \tau_{\text{M}}(x, b_{\mathcal{Y}}, a, o, x')$
   through sawtooth approximation [7] over the points in $\Upsilon_{\mathcal{Y}}(x')$.
2: For all $a \in \mathcal{A}$, $\overline{Q}(x, b_{\mathcal{Y}}, a) = \sum_y b_{\mathcal{Y}}(y) R(x, y, a) + \gamma \sum_{y,o,x',y'} (b_{\mathcal{Y}}(y) T_{\mathcal{X}}(x, y, a, x')$
       $\times T_{\mathcal{Y}}(x, y, a, x', y') Z(x', y', a, o)\, \overline{v}_{a,o,x'})$.
3: $\overline{v} = \max_a \overline{Q}(x, b_{\mathcal{Y}}, a)$.
4: Insert $(b_{\mathcal{Y}}, \overline{v})$ into $\Upsilon_{\mathcal{Y}}(x)$.
```

PRUNE. Invocation of SAMPLE and BACKUP generates new nodes in $T_{\mathcal{R}}$ and α-vectors. However, not all of them are useful for constructing an optimal policy and are pruned for computational efficiency. To prune belief points, we compare the upper bound on the value for taking an action a at a belief node b with the lower bound on the value for taking some other action a' at b. If it is smaller, then taking action a at b is clearly suboptimal, and we prune all the nodes of the subtree associated with a. To prune α-vectors, we go through each α-vector set in the collection $\{\Gamma_{\mathcal{Y}}(x) \mid x \in \mathcal{X}\}$ and prune any α-vector in $\Gamma_{\mathcal{Y}}(x)$ that does not dominate the rest at some sampled point $(x, b_{\mathcal{Y}})$, where $b_{\mathcal{Y}} \in \mathcal{B}_{\mathcal{Y}}(x)$.

Our description of the algorithm is quite brief, as the main purpose is to illustrate the use of the MOMDP representation. More details and justifications for the particular choices of the sampling, backup, and pruning strategies can be found in [14].

4.3 Convergence

The SARSOP algorithm's convergence property relies on sampling the belief search tree $T_{\mathcal{R}}$ adequately to a sufficient depth, performing backups to improve the upper and lower bounds at the nodes along the sampled paths, and propagating the improvement back to the initial belief at the root of the tree. To solve a MOMDP, the main modifications required for SARSOP are the belief update equation (8) and the backup operations on the lower bound (Algorithm 2) and the upper bound (Algorithm 3). These primitive operations do not affect the convergence property of SARSOP, and the new algorithm provides the same theoretical guarantee as the original SARSOP algorithm [14].

To see this informally, the belief update operation under the MOMDP representation (Eq. (8)) is equivalent to that in SARSOP (Eq. (2)). Modeling a task as a MOMDP does not change its belief-space dynamics. The beliefs reached after any arbitrary sequence of actions and observations, in the corresponding MOMDP and POMDP, are equivalent. This can be shown formally using the same technique as that in the proof of Theorem 1 (see Appendix A).

The main difference in the new algorithm's backup operation lies in the lower bound backup. Algorithm 2 shows that lower bound backup at a belief $(x, b_{\mathcal{Y}})$ generates a new α-vector that spans only the subspace $\mathcal{B}_{\mathcal{Y}}(x)$ and thus can improve the lower bound for beliefs in $\mathcal{B}_{\mathcal{Y}}(x)$, but not for beliefs in other subspaces $\mathcal{B}_{\mathcal{Y}}(x')$ with $x' \neq x$. In contrast, the backup operation in the original SARSOP algorithm generates an α-vector that spans the entire space $\mathcal{B}$, and thus its *potential* to improve the lower bound is not restricted to any particular subspace of $\mathcal{B}$. However, SARSOP's convergence property is not contingent on guaranteeing lower bound improvement everywhere in $\mathcal{B}$ during the backup operation. In fact, such a guarantee is not possible for any point-based POMDP algorithm. Thus SARSOP's convergence property is unaffected and holds for the new algorithm for MOMDPs as well.

4.4 Computational Efficiency

MOMDPs allow the belief space $\mathcal{B}$ to be represented as a union of low-dimensional subspaces $\mathcal{B}_{\mathcal{Y}}(x)$ for $x \in \mathcal{X}$. To understand the resulting computational advantages, let us compare the key primitive operations—belief update, backup, and pruning—under the standard POMDP model and the MOMDP model. We assume an efficient sparse vector representation for the beliefs and a sparse matrix representation for the transition and observation functions whenever possible.

In a mixed observability task, belief vectors under the POMDP model contain many zero entries and can be processed and stored efficiently under a sparse representation. The MOMDP model does not bring further advantage for belief update.

The efficiency gain of our algorithm comes mainly from faster backup and pruning operations. In a MOMDP, α-vectors span the subspace $\mathcal{B}_{\mathcal{Y}}(x)$ and have length $|\mathcal{Y}|$, while in the corresponding POMDP, α-vectors span $\mathcal{B}$ and have length $|\mathcal{X}| \times |\mathcal{Y}|$. The backup operation for constructing the α-vectors (Algorithm 2, line 2) is thus faster by a factor of $|\mathcal{X}|$ in our algorithm. The pruning operation also becomes more effective. In a standard POMDP, an α-vector can be pruned only if it is dominated over the entire $\mathcal{B}$. In a MOMDP, an α-vector can be pruned if it is dominated over a subspace $\mathcal{B}_{\mathcal{Y}}(x)$, which happens a lot more frequently. Thus our algorithm prunes α-vectors more aggressively.

The efficiency gain in BACKUP-LB sometimes comes at a cost: although α-vectors in a MOMDP are shorter, they also contain less information, compared with α-vectors in the corresponding POMDP. MOMDP backup at $(x, b_{\mathcal{Y}})$ generates an α-vector that spans only the subspace $\mathcal{B}_{\mathcal{Y}}(x)$ and does not generate information in any other subspace $\mathcal{B}_{\mathcal{Y}}(x')$ with $x' \neq x$. In contrast, POMDP backup does more computation and generates an α-vector that spans the entire space $\mathcal{B}$. If a problem has many similar fully observable states in the sense that the α-vectors in one belief subspace $\mathcal{B}_{\mathcal{Y}}(x)$ are useful in many other subspaces $\mathcal{B}_{\mathcal{Y}}(x')$ with $x \neq x'$, 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. This, however, requires a special property which does not hold in general for complex systems. An α-vector which is optimal at some belief in $\mathcal{B}_{\mathcal{Y}}(x)$ may not be optimal at any belief in $\mathcal{B}_{\mathcal{Y}}(x')$ with $x \neq x'$. In this case, the extra computation that POMDP backup performs is superfluous, as the result does not form part of the optimal solution. This is especially true, when a system visits only a small fraction of the entire state space under an optimal policy. An example is shown in Figure 6.

We would also like to point out that the belief update and backup operations, which are the main modifications required for SARSOP to solve MOMDPs, are common to most point-based POMDP algorithms, such as PBVI [16], Perseus [27], and HSVI [25]. Replacing these operations with corresponding ones introduced here would allow these algorithms to benefit from the efficiency gain of the MOMDP approach just as SARSOP does.

5 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.

5.1 Robotic Tasks

Tag. The Tag problem first appeared in the work on PBVI [16], 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 catches the target, *i.e.*, arrives in the same position as that of the target.

In the MOMDP for this task, the fully observable state variable x models the robot position, which is known exactly. The x variable also takes one extra value that indicates that the robot and the target are in the same position. The partially observable state variable y models the target position, 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. Thus we have $|\mathcal{X}| = M + 1$ and $|\mathcal{Y}| = M$, while in the standard POMDP model, the state space has $|\mathcal{S}| = (M + 1)M$ dimensions.

Two-Robot Tag. This task is a variation of Tag. Two robots attempt to catch a target, which always moves away from the closer robot (Figure 5). We assume centralized planning and execution. The two robots maintain a communication link and share their knowledge of actions and observations. The main objective here is to test our algorithm's ability to handle more than one robot. Similar to Tag, the x variable in the MOMDP models the two robots' positions, and the y variable models the target position. TwoRobotTag(M) denotes an experiment on a map with M positions.

Rock Sample. The Rock Sample problem [25] has frequently been used to test the scalability of new POMDP algorithms. It models a planetary rover exploring an area represented as a grid and searching 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 has scientific value.

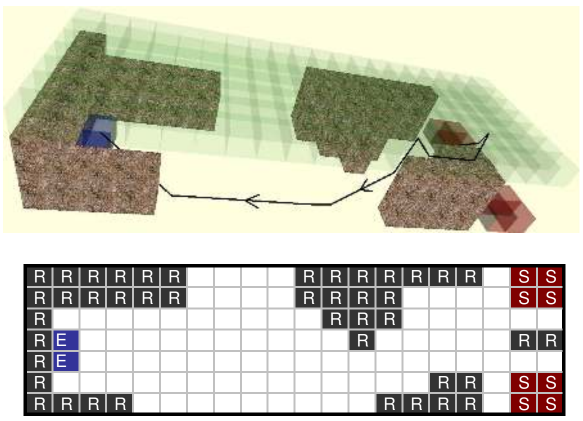

Figure 4: AUV navigation. Top: The 3-D environment and an AUV simulation trajectory (marked in black) generated from the computed policy. The AUV can localize its horizontal position only at the surface level (shaded in light green). The simulation trajectory shows that the AUV rises to the surface level 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.

Here, the x variable in the MOMDP represents the robot position, and the y variable is a binary vector in which each element indicates whether a particular rock has scientific value or not. Experiments were performed on maps of different sizes and with different numbers of rocks. RockSample(M, R) denotes a map size of $M \times M$ and R rocks.

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 (Figure 4). 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. The orientation 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.

It may be tempting to approximate the belief on the AUV state as a Gaussian distribution. However, due to significant uncertainty on the AUV’s initial state, the belief is in fact not even unimodal (Figure 4), and Gaussian approximation likely causes too much loss in accuracy. The MOMDP model handles beliefs of general form and does not make such approximations.

5.2 Results

We applied the MOMDP algorithm described in Section 4 to the four tasks. 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 two hours to reach this level or until

Table 1: 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
TwoRobotTag(24)			
$\lvert\mathcal{X}\rvert$=625,$\lvert\mathcal{Y}\rvert$=24	MOMDP	-12.07 ± 0.15	1636
$\lvert\mathcal{S}\rvert$=14,400, $\lvert\mathcal{A}\rvert$=25, $\lvert\mathcal{O}\rvert$=625		-11.37 ± 0.15	1831
	SARSOP	-12.10 ± 0.16	4095
		-12.09 ± 0.16	7200
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	21.47 ± 0.04	1061
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	21.57 ± 0.04	103
$\lvert\mathcal{S}\rvert$=247,809, $\lvert\mathcal{A}\rvert$=16, $\lvert\mathcal{O}\rvert$=2		22.48 ± 0.03	1879
	SARSOP	21.56 ± 0.11	1369*
RockSample(12,11)			
$\lvert\mathcal{X}\rvert$=145,$\lvert\mathcal{Y}\rvert$=2,048	MOMDP	21.45 ± 0.04	24
$\lvert\mathcal{S}\rvert$=294,913, $\lvert\mathcal{A}\rvert$=16, $\lvert\mathcal{O}\rvert$=2		22.00 ± 0.04	836
	SARSOP	21.38 ± 0.04	963*
AUV Navigation			
$\lvert\mathcal{X}\rvert$=96,$\lvert\mathcal{Y}\rvert$=141	MOMDP	799.9 ± 3.5	112
$\lvert\mathcal{S}\rvert$=13,536, $\lvert\mathcal{A}\rvert$=6, $\lvert\mathcal{O}\rvert$=144		808.0 ± 3.4	123
	SARSOP	799.3 ± 2.9	3463
		795.7 ± 3.5	7200

*The program ran out of memory.

the program ran out of memory. 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. For comparison, we also ran the original SARSOP algorithm [14] on the same tasks modeled as standard POMDPs. Both algorithms are implemented in C++. The implementation uses an efficient sparse vector representation for the beliefs and a sparse matrix representation for the transition and observation functions. The experiments were performed on a PC with a 2.66GHz Intel processor and 2GB memory.

5.2.1 Mixed Observability Tasks

The results are shown in Table 1. Column 1 of the table lists each task, its state space size $|\mathcal{S}|$ for the standard POMDP model, $|\mathcal{X}|$ and $|\mathcal{Y}|$ for the MOMDP model, as well as action space and observation space sizes, $|\mathcal{A}|$ and $|\mathcal{O}|$. Small optimizations can be performed to reduce the number of states in the POMDP model. So $|\mathcal{S}|$ may not be exactly equal to $|\mathcal{X}| \times |\mathcal{Y}|$ for the corresponding MOMDP. However, this is not significant enough to affect $|\mathcal{S}| = O(|\mathcal{X}||\mathcal{Y}|)$. Column 3 of the table lists the estimated expected total rewards for the computed

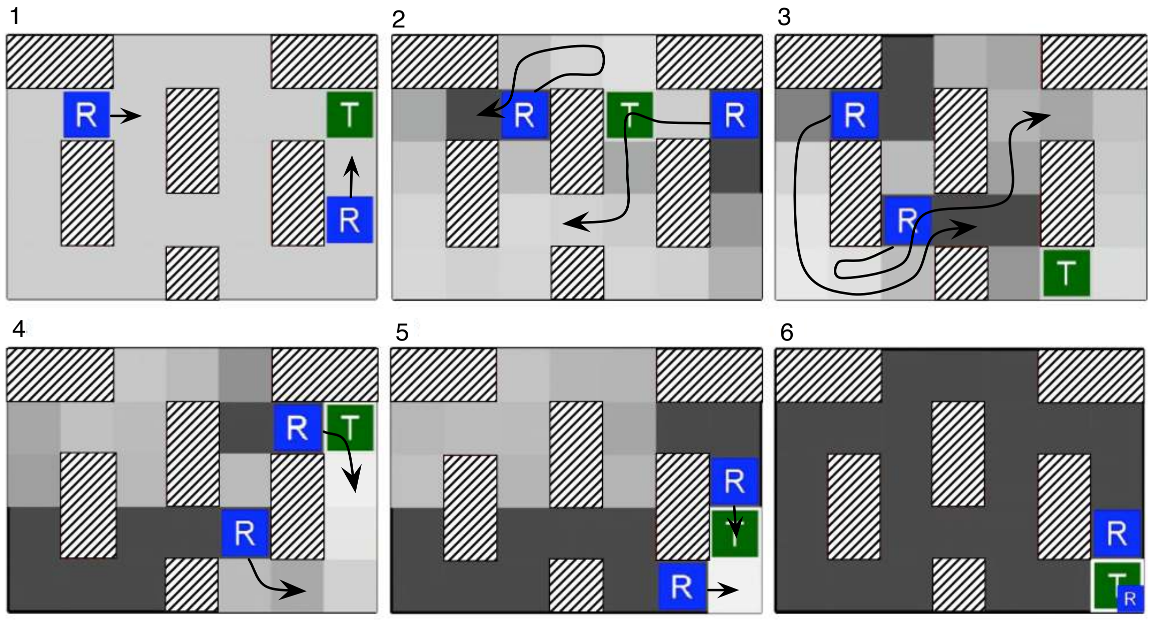

Figure 5: A simulation trajectory under a policy for Two-Robot Tag. Striped regions indicate obstacles. The two "R"s indicate the robots' positions, and the "T" indicates the target's position. The various shades of gray indicate the belief on the target position. Lighter colors indicate higher probabilities, with dark gray indicating zero probability.

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 experiments on Tag with two different map sizes. 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, increases as well. We also ran multiple experiments for Rock Sample, with different map sizes and numbers of rocks, and found a similar trend. In particular, for the largest problems, RockSample(11,11) and RockSample(12,11), which have 250,000 and 300,000 states respectively, SARSOP never reached the same reward level attained by the MOMDP algorithm before running out of memory. This is not surprising as the computational efficiency gain achievable from the MOMDP model increases with $|\mathcal{X}|$ (Section 4.4). For both TwoRobotTag(24) and AUV Navigation, SARSOP also failed to reach the same reward level attained by the MOMDP algorithm.

Figure 5 shows snapshots of a single simulation run of a policy that our algorithm computed for TwoRobotTag(24). The robots' initial belief on the target position is uniform over all possible positions, as there is no prior information (snapshot 1). The two robots then move toward the top middle part of the environment to search for the target (snapshots 1 and 2), while updating the belief using the observations received and the model of target dynamics. The belief is now non-uniform with several high-probability regions. Next, the robots move to search the lower left corner (snapshot 3) and lower right corner (snapshot 4) in turn. It is interesting to note that in both cases, the two robots coordinate to approach the corner from opposite sides of the obstacle, in order to surround the target, if it tries to escape. Also, as the robots move from the lower left to lower right corner, they take as short paths as possible (snapshot 3), because there is cost for each move. Throughout the run, prior to the target's capture, the robots never see the target, and yet right before the capture, the robots have eliminated the possibility of the target hiding in a large portion of the environment (the dark gray area in snapshot 5), but not all. They finally capture the target in the lower right corner (snapshot 6). If the robots do not capture the

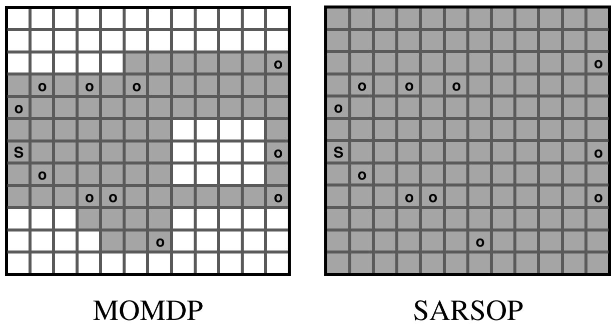

Figure 6: RockSample(12,11). “S” marks the initial robot position, “o” marks rock positions. The robot’s final destination is on the right side of the map. Shaded cells represent robot positions for which the algorithms perform backup and generate α-vectors.

target there, it is expected that they will search the top left part of the environment again.

To understand the computational advantages of the MOMDP algorithm, let us look at Rock Sample more closely. In this task, x represents the robot position, and y represents the value of rocks. For each position x, $\Gamma_{\mathcal{Y}}(x)$ is a set of α-vectors that specify the actions for the robot if it is located at x and has some belief $b_{\mathcal{Y}}$ on the rocks. Suppose that under an optimal policy, the robot never visits a position x', starting from its initial position. Then there is no need to construct the α-vector set $\Gamma_{\mathcal{Y}}(x')$. The MOMDP algorithm automatically benefits from such computational efficiency gains. Figure 6 shows a concrete example. After 836 seconds of policy computation, the MOMDP algorithm obtains a better policy than the SARSOP algorithm. Yet, the MOMDP algorithm performs backup and generates α-vectors in only roughly 60% of robot positions, while the SARSOP algorithm, which uses a standard POMDP model, generates α-vectors for all robot positions. We believe that this is an important contributor to the efficiency of the MOMDP algorithm.

5.2.2 Pseudo Full Observability and Reparameterized Full Observability

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, following the pseudo full observability technique in Section 3.3.1, we modeled the task as a MOMDP by assuming that the robot position is fully observable. We then averaged over the value function computed by the MOMDP algorithm and generated a policy to control the robot under the original POMDP model, without assuming any fully observable state variables for policy execution.

We ran experiments with different sensor accuracy p for the Noisy Tag problem. The results are shown in Table 2. 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 loses to SARSOP in the long run. However, in this case we did not observe any significant performance loss for the MOMDP policy. To confirm the results, we ran SARSOP for two additional hours, but the reward level of the resulting policy did not improve much beyond those reported in the table.

To examine the reparameterized full observability technique, the Tag problem is modified so that the robot

Table 2: Performance comparison on tasks using pseudo or reparameterized full observability techniques.

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

never observes its own position exactly, but only the 3×3 region around it in the grid. Now both the robot and the target positions are partially observable. However, the preimage of any observation on the robot position is bounded. We can apply the approach described in Section 3.3.2 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 outperformed SARSOP (Table 2). This suggests that extracting fully observable state variables through reparameterization is a promising idea and deserves further investigation.

6 Software Implementation

We implemented the MOMDP algorithm in a C++ software package APPL, which stands for *Approximate POMDP Planning Library*. It is 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 and flexible way. APPL is backward compatible and can read the standard POMDP format as well. As an example, the file for the benchmark problem RockSample(7,8) has 18.9 MB in the standard format, but only 0.1 MB in the new POMDPX format.

- APPL also allows POMDP/MOMDP models to be specified in C++ and integrated with the solver algorithm through a well-defined programming API.

- APPL provides several ways to examine the computed policy. It provides 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.

7 Conclusion

POMDPs have been successfully used for motion planning under uncertainty in various robotic tasks [10, 16, 17, 26]. 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. The factored representation drastically improves the speed of POMDP planning, when combined with a point-based POMDP algorithm. Furthermore, even when a robot does not have obvious fully observable state components, it still can sometimes be modeled as a MOMDP by reparameterizing the robot's state space. Experiments show that on a set of test problems, our new approach improves the performance of a leading point-based POMDP algorithm by many times.

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 achievement. 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 forward in scaling up POMDP algorithms and ultimately making them practical for robot motion planning in uncertain and dynamic environments.

Acknowledgments. The authors would like to 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, Singapore.

References

[1] J.A. Boyan and M.L. Littman. Exact solutions to time-dependent MDPs. In *Advances in Neural Information Processing Systems (NIPS)*, 2000.

[2] A. Brooks, A. Makarendo, S. Williams, and H. Durrant-Whyte. Parametric POMDPs for planning in continuous state spaces. *Robotics & Autonomous Systems*, 54(11):887–897, 2006.

[3] C. Guestrin, M. Hauskrecht, and B. Kveton. Solving factored MDPs with continuous and discrete variables. In *Proc. Uncertainty in Artificial Intelligence*, pages 235–242, 2004.

[4] 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*, pages 67–75, 2001.

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

[6] E.A. Hansen and Z. Feng. Dynamic programming for POMDPs using a factored state representation. In *Proc. Int. Conf. on AI Planning Systems*, pages 130–139, 2000.

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

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

[9] J. Hoey, A. von Bertoldi, P. Poupart, and A. Mihailidis. Assisting persons with dementia during handwashing using a partially observable Markov decision process. In *Proc. Int. Conf. on Vision Systems*, 2007.

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

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

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

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

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

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

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

[17] 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.

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

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

[20] N. Roy and S. Thrun. Coastal navigation with mobile robots. In *Advances in Neural Information Processing Systems (NIPS)*, volume 12, pages 1043–1049, 1999.

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

[22] G. Shani, P. Poupart, R.I. Brafman, and S.E. Shimony. Efficient ADD operations for point-based algorithms. In *Proc. Int. Conf. on Automated Planning & Scheduling*, 2008.

[23] H.S. Sim, K.E. Kim, J.H. Kim, D.S. Chang, and M.W. Koo. Symbolic heuristic search value iteration for factored POMDPs. In *Proc. Nat. Conf. on Artificial Intelligence*, pages 1088–1093, 2008.

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

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

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

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

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

[29] J.D. Williams, P. Poupart, and S. Young. Factored partially observable Markov decision processes for dialogue management. In *IJCAI Workshop on Knowledge & Reasoning in Practical Dialog Systems*, 2005.

A Proof of Theorem 1

We start by defining the equivalence between a POMDP belief and a MOMDP belief for a system whose state s consists of a fully observable component x and a partially observable component y. For the POMDP model $(\mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R, \gamma)$, a belief b on s is a probability distribution over $\mathcal{S}$. Since the state variable x is fully observable, b has a special structure: if x has value ξ, then $b(x, y) = 0$ for all $x \neq \xi$ and all y. For the corresponding reparameterized MOMDP model $(\mathcal{X}, \mathcal{Y}, \mathcal{A}, \mathcal{O}, T_{\mathcal{X}}, T_{\mathcal{Y}}, Z_{\mathrm{M}}, R_{\mathrm{M}}, \gamma)$ with $\mathcal{X} = \mathcal{O}$, the belief $b_{\mathcal{Y}}$ on y is a probability distribution over $\mathcal{Y}$. We say that a POMDP belief b whose observed state variable x has value ξ and a MOMDP belief $(\xi, b_{\mathcal{Y}})$ are *equivalent* if $b(s) = b_{\mathcal{Y}}(y)$ for all $s = (\xi, y)$.

The first statement of Theorem 1 claims that given equivalent initial beliefs, the beliefs reached under a POMDP model and the reparameterized MOMDP model are equivalent after any arbitrary sequence of actions and observations. We prove this by induction on the length t of the action-observation sequence.

Proof. The base case $t = 0$ holds trivially, as the initial beliefs are equivalent. Now assume that a POMDP belief b^t and a MOMDP belief $(x^t, b^t_{\mathcal{Y}})$ are equivalent after any arbitrary action-observation sequence of length t, *i.e.*, $b^t(s) = b^t_{\mathcal{Y}}(y)$ for all $s = (x^t, y)$. Consider the belief b^{t+1} reached after an action-observation sequence of length $t+1$ under the POMDP model and similarly the belief $(x^{t+1}, b^{t+1}_{\mathcal{Y}})$ under the MOMDP model. Let a and o denote the $(t+1)$'th action and observation. We have

$$\begin{aligned} b^{t+1}_{\mathcal{Y}}(y') &= \eta Z_{\mathrm{M}}(x^{t+1}, y', a, o) \sum_{y \in \mathcal{Y}} T_{\mathcal{X}}(x^t, y, a, x^{t+1}) T_{\mathcal{Y}}(x^t, y, a, x^{t+1}, y') b^t_{\mathcal{Y}}(y) \\ &= \eta \sum_{y \in \mathcal{Y}} T_{\mathcal{X}}(x^t, y, a, x^{t+1}) T_{\mathcal{Y}}(x^t, y, a, x^{t+1}, y') b^t_{\mathcal{Y}}(y) \\ &= \eta \sum_{y \in \mathcal{Y}} T(s, a, s') Z(s', a, o) b^t_{\mathcal{Y}}(y), \end{aligned}$$

where $s = (x^t, y)$ and $s' = (x^{t+1}, y')$. The first line above follows from (8), the second line follows because $x^{t+1} = o$ for the MOMDP by construction, and the third line follows from the definition of $T_{\mathcal{Y}}$ in (5). By the induction hypothesis, $b^t(s) = b^t_{\mathcal{Y}}(y)$ for all $s = (x^t, y)$. Furthermore, $b^t(s) = 0$ for all $s = (x, y)$ with $x \neq x^t$, because the state variable x is fully observable. It then follows that for all $s' = (x^{t+1}, y')$,

$$b^{t+1}_{\mathcal{Y}}(y') = \eta Z(s', a, o) \sum_{s \in \mathcal{S}} T(s, a, s') b^t(s) = b^{t+1}(s').$$

□

Next we consider the second statement of Theorem 1. Given a policy π, let $V^t(b)$ be the expected total reward after executing π for t steps starting with an initial belief b under the POMDP model. Similarly let

$V^t_{\mathrm{M}}(x, b_{\mathcal{Y}})$ be the expected total reward after executing π for t steps starting with an initial belief $(x, b_{\mathcal{Y}})$ under the MOMDP model. We want to show that if b and $(x, b_{\mathcal{Y}})$ are equivalent beliefs, then $V^t(b) = V^t_{\mathrm{M}}(x, b_{\mathcal{Y}})$ for all t, again by induction.

Proof. The base case of $V^0(b) = V^0_{\mathrm{M}}(x, b_{\mathcal{Y}}) = 0$ when b and $(x, b_{\mathcal{Y}})$ are equivalent holds trivially. Now suppose that $V^t(b) = V^t_{\mathrm{M}}(x, b_{\mathcal{Y}})$ for any POMDP belief b and MOMDP belief $(x, b_{\mathcal{Y}})$ that are equivalent. Consider $V^{t+1}(b)$ and $V^{t+1}_{\mathrm{M}}(x, b_{\mathcal{Y}})$:

$$V^{t+1}(b) = \sum_{s\in\mathcal{S}} b(s)R(s,a) + \gamma \sum_{o\in\mathcal{O}} p(o|b,a)V^t(b'), \tag{9}$$

with $a = \pi(b)$ and $b' = \tau(b, a, o)$;

$$V^{t+1}_{\mathrm{M}}(x, b_{\mathcal{Y}}) = \sum_{y\in\mathcal{Y}} b_{\mathcal{Y}}(y)R_{\mathrm{M}}(x, y, a_{\mathrm{M}}) + \gamma \sum_{x'\in\mathcal{X}, o\in\mathcal{O}} p(x', o|x, b_{\mathcal{Y}}, a_{\mathrm{M}})V^t_{\mathrm{M}}(x', b'_{\mathcal{Y}}), \tag{10}$$

with $a_{\mathrm{M}} = \pi(x, b_{\mathcal{Y}})$ and $b'_{\mathcal{Y}} = \tau_{\mathrm{M}}(x, b_{\mathcal{Y}}, a_{\mathrm{M}}, o, x')$. We now show that the corresponding terms in (9) and (10) are equal. First, by definition, $R_{\mathrm{M}}(x, y, a) = R(s, a)$ where $s = (x, y)$, and for any a. Since b and $(x, b_{\mathcal{Y}})$ are equivalent beliefs, we have $a = \pi(b) = \pi(x, b_{\mathcal{Y}}) = a_{\mathrm{M}}$ and

$$\sum_{y\in\mathcal{Y}} R_{\mathrm{M}}(x, y, a_{\mathrm{M}})b_{\mathcal{Y}}(y) = \sum_{s\in\mathcal{S}} R(s,a)b(s). \tag{11}$$

Next, using the first statement of the theorem, we conclude that b' and $(x', b'_{\mathcal{Y}})$ are equivalent. Combining this with the induction hypothesis, we have $V^t_{\mathrm{M}}(x', b'_{\mathcal{Y}}) = V^t(b')$. For the reparameterized MOMDP, $x' = o$, and thus

$$V^t_{\mathrm{M}}(o, b'_{\mathcal{Y}}) = V^t(b'). \tag{12}$$

Finally,

$$\begin{aligned} p(x', o|x, b_{\mathcal{Y}}, a_{\mathrm{M}}) &= \sum_{y\in\mathcal{Y}, y'\in\mathcal{Y}} p(x', y', o|x, y, a_{\mathrm{M}})b_{\mathcal{Y}}(y) \\ &= \sum_{y\in\mathcal{Y}, y'\in\mathcal{Y}} T_{\mathcal{X}}(x, y, a_{\mathrm{M}}, x')T_{\mathcal{Y}}(x, y, a_{\mathrm{M}}, x', y')Z_{\mathrm{M}}(x', y', a_{\mathrm{M}}, o)b_{\mathcal{Y}}(y), \end{aligned}$$

and

$$\begin{aligned} \sum_{x'\in\mathcal{X}, o\in\mathcal{O}} p(x', o|x, b_{\mathcal{Y}}, a_{\mathrm{M}})V^t_{\mathrm{M}}(x', b'_{\mathcal{Y}}) &= \sum_{o\in\mathcal{O}} \sum_{y\in\mathcal{Y}, y'\in\mathcal{Y}} T_{\mathcal{X}}(x, y, a_{\mathrm{M}}, o)T_{\mathcal{Y}}(x, y, a_{\mathrm{M}}, o, y')b_{\mathcal{Y}}(y)V^t_{\mathrm{M}}(o, b'_{\mathcal{Y}}) && (13) \\ &= \sum_{o\in\mathcal{O}} \sum_{s\in\mathcal{S}, s'\in\mathcal{S}} T(s, a, s')Z(s', a, o)b(s)V^t(b') && (14) \\ &= \sum_{o\in\mathcal{O}} p(o|b,a)V^t(b') && (15) \end{aligned}$$

Eq. (13) follows, because $Z_{\mathrm{M}}(x', y', a_{\mathrm{M}}, o)$ is 1 if $x' = o$ and 0 otherwise. Eq. (14) uses the definitions of $T_{\mathcal{X}}$ and $T_{\mathcal{Y}}$, the fact that $(x, b_{\mathcal{Y}})$ and b are equivalent beliefs, and (12). Substituting (11) and (15) into (10), we get the desired results. □