Robotics: Science and Systems 2008
Zurich, CH, June 25-28, 2008

SARSOP: Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces

Hanna Kurniawati David Hsu Wee Sun Lee
Department of Computer Science, National University of Singapore
Singapore 117590, Singapore

***Abstract*— Motion planning in uncertain and dynamic environments is an essential capability for autonomous robots. Partially observable Markov decision processes (POMDPs) provide a principled mathematical framework for solving such problems, but they are often avoided in robotics due to high computational complexity. Our goal is to create practical POMDP algorithms and software for common robotic tasks. To this end, we have developed a new point-based POMDP algorithm that exploits the notion of *optimally reachable belief spaces* to improve computational efficiency. In simulation, we successfully applied the algorithm to a set of common robotic tasks, including instances of coastal navigation, grasping, mobile robot exploration, and target tracking, all modeled as POMDPs with a large number of states. In most of the instances studied, our algorithm substantially outperformed one of the fastest existing point-based algorithms. A software package implementing our algorithm is available for download at `http://motion.comp.nus.edu.sg/projects/pomdp/pomdp.html`.**

I. Introduction

Partially observable Markov decision processes (POMDPs) [17] provide a principled mathematical framework for planning under uncertainty, an essential capability for robots operating in uncertain and dynamic environments. However, POMDPs are often avoided in robotics, because solving POMDPs exactly is computationally intractable [9]. Not long ago, the best algorithms could spend hours computing exact solutions to POMDPs with only a dozen states, which are woefully inadequate for modeling realistic robotic tasks. In recent years, point-based POMDP algorithms [5, 10, 16, 19, 20] have made impressive progress by computing good approximate solutions: POMDPs with hundreds of states have been solved in a matter of seconds (*e.g.*, [5, 16, 19]). These algorithms have the potential to make POMDPs practical for many applications in robotics and beyond.

Our goal is to create practical POMDP algorithms and software for common robotic tasks. To this end, we have developed a new point-based POMDP algorithm that exploits the notion of *optimally reachable belief spaces* to improve computational efficiency. In simulation, we successfully applied our algorithm to a set of common robotic tasks, including coastal navigation, grasping, mobile robot exploration, and target tracking, all modeled as POMDPs with a large number of states.

POMDP algorithms typically operate in a robot's *belief space*. A *belief* is a probability distribution over all possible robot states, and the set of all beliefs form the belief space. Intuitively, the difficulty of solving POMDPs is due to the "curse of dimensionality": in a discrete POMDP, 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, robot navigation in a simple planar environment modeled as a 10×10 grid. The resulting belief space is 100-dimensional!

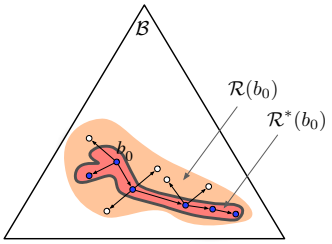


Fig. 1. Belief space $\mathcal{B}$, reachable space $\mathcal{R}(b_0)$, and optimally reachable space $\mathcal{R}^*(b_0)$. Note that $\mathcal{R}^*(b_0) \subseteq \mathcal{R}(b_0) \subseteq \mathcal{B}$.

To overcome this difficulty, one key idea of point-based POMDP algorithms is to *sample* a set of points from $\mathcal{B}$ and use it as an approximate representation of $\mathcal{B}$, instead of representing $\mathcal{B}$ exactly. Some early POMDP algorithms sample the entire belief space $\mathcal{B}$, using a uniform sampling distribution, such as a grid. However, it is difficult to sample a representative set of points from $\mathcal{B}$, due to its large size. More recent point-based algorithms sample only $\mathcal{R}(b_0)$, the subset of belief points reachable from a given initial point $b_0 \in \mathcal{B}$, under arbitrary sequences of actions (Fig. 1). It is generally believed that $\mathcal{R}(b_0)$ is much smaller than $\mathcal{B}$. Indeed, focusing on $\mathcal{R}(b_0)$ allows point-based algorithms to scale up to larger problems. To push further in this direction, we would like to sample near $\mathcal{R}^*(b_0)$, the subset of belief points reachable from b_0 under *optimal* sequences of actions, as $\mathcal{R}^*(b_0)$ is usually much smaller than $\mathcal{R}(b_0)$. Of course, the optimal sequences of actions constitute exactly the POMDP solution, which is unknown in advance. In fact, knowing $\mathcal{R}^*(b_0)$ is in some sense "equivalent" to knowing the POMDP solution (see Section III-A). So we need to approximate $\mathcal{R}^*(b_0)$.

The main idea of our algorithm is to compute successive approximations of $\mathcal{R}^*(b_0)$ and converge to it iteratively. Since $\mathcal{R}^*(b_0)$ is unknown in advance, the algorithm relies on heuristic exploration to sample $\mathcal{R}(b_0)$ and improves sampling over time through a simple on-line learning technique. It then uses a bounding technique to avoid sampling in regions that are unlikely to be optimal and focus sampling on the region near $\mathcal{R}^*(b_0)$, the subset of $\mathcal{B}$ most relevant to the POMDP solution. This leads to substantial gain in computational efficiency.

Focusing on $\mathcal{R}^*(b_0)$ also brings an indirect benefit. Under fairly general conditions, the solution to a POMDP can be represented as a convex, piecewise-linear *value function* [17].

We represent the value function as a set Γ of hyperplanes, each of which must dominate the rest at some sampled point. By pruning away sampled points that are suboptimal, *i.e.*, outside $\mathcal{R}^*(b_0)$, we can reduce the size of Γ, thus further improving computational efficiency.

II. Background

A. POMDPs

A POMDP models an agent taking a sequence of actions under uncertainty to maximize its reward. Formally it 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 state. Due to the uncertainty in observation, the observation result $o \in \mathcal{O}$ is again modeled as a conditional probability function $Z(s, a, o) = p(o|s, a)$.

In each step, the agent receives a real-valued reward $R(s, a)$, if it takes action a in state s, and the goal of the agent 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 given by $\mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$, where s_t and a_t denote the agent's state and action at time t.

The solution to a POMDP is an *optimal policy* that maximizes the expected total reward. Normally, a policy is a mapping from the agent's state to a prescribed action. However, in a POMDP, the agent's state is partially observable and not known exactly. So we rely on the concept of beliefs. As described earlier, a belief is a probability distribution over $\mathcal{S}$. A POMDP policy $\pi\colon \mathcal{B} \to \mathcal{A}$ maps a belief $b \in \mathcal{B}$ to a prescribed action $a \in \mathcal{A}$.

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

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

where Γ is a finite set of vectors called α-vectors, b is the discrete vector representation of a belief, and $\alpha \cdot b$ is the inner product of vectors α-vector and b. Each α-vector is associated with an action. The policy can be executed by selecting the action corresponding to the best α-vector at the current belief. So a policy can be represented as a set of α-vectors.

B. Related Work

POMDPs are a principled approach for planning and decision making under uncertainty [6, 17], but they are notoriously hard to solve [7, 9]. There have been significant efforts in developing approximation algorithms. See [1] for a recent survey.

Point-based algorithms have been particularly successful in computing approximate solutions to large POMDPs [2, 5, 10, 16, 19, 20]. Most of them use *value iteration* [13]. Exploiting the fact that the optimal value function must satisfy the Bellman equation [13], value iteration algorithms start with an initial policy represented as a value function V and perform backup operations on V by iterating on the Bellman equation until the iteration converges. One important idea shared by the point-based algorithms is to sample a representative set of points from the belief space $\mathcal{B}$ and compute an approximately optimal value function by performing backup operations over the sampled points rather than the entire $\mathcal{B}$. They differ in how they sample the belief space and perform backup operations. To improve computational efficiency, recent point-based algorithms sample only the reachable space $\mathcal{R}(b_0)$ from an initial belief point b_0.

PBVI [10] is the first point-based algorithm that demonstrated good performance on a large POMDP called Tag, which has 870 states. Later point-based algorithms improved the performance significantly on this and other larger POMDPs. To our knowledge, HSVI2 [19] so far has the best performance in general. HSVI2 uses heuristics to guide the sampling towards regions that help cut down the gap between the upper and lower bounds on the optimal value function. FSVI [16] is another point-based algorithm, which uses a Markov decision process (MDP) to guide the sampling. MDP-guided sampling is effective for some problems, but the performance degrades when uncertainty is high and long sequences of information-gathering actions are required.

Our algorithm is related to HSVI2 and FSVI, but it explicitly attempts to sample the optimally reachable space $\mathcal{R}^*(b_0)$ through learning-enhanced exploration and a bounding technique. Experimental results show that focusing on $\mathcal{R}^*(b_0)$ is a promising idea. An early version of our algorithm [5] exploits bounding in a limited way: bounds are compared locally at individual belief points to prune suboptimal actions. In contrast, the current algorithm sets up the bounds to reach a specified value function approximation level at b_0, thereby leveraging information globally to reduce the number of poor samples—those that are in $\mathcal{R}(b_0)$ but not in $\mathcal{R}^*(b_0)$.

One crucial reason for the computational intractability of POMDPs is the high dimensionality of $\mathcal{B}$. Low-dimensional approximations of $\mathcal{B}$ therefore improve computational efficiency greatly (*e.g.*, [12, 14]). These approaches are important, but beyond the scope of this paper.

III. SARSOP

We now describe our algorithm, SARSOP, which stands for Successive Approximations of the Reachable Space under Optimal Policies.

A. Optimally Reachable Spaces

A key idea of point-based POMDP algorithms is to sample a representative set of points from the belief space and

Algorithm 1 SARSOP.

1: Initialize the set Γ of α-vectors, 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 b_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 b, BACKUP($T_{\mathcal{R}}, \Gamma, b$).
6: PRUNE($T_{\mathcal{R}}$, Γ).
7: **until** termination conditions are satisfied.
8: **return** Γ.

use it as an approximate representation of the space. For efficiency, most recent algorithms sample from $\mathcal{R}(b_0)$, the set of points reachable from a given point $b_0 \in \mathcal{B}$ under arbitrary sequences of actions. Theoretical analysis shows that approximate POMDPs solutions can be computed efficiently, when $\mathcal{R}(b_0)$ has a small *covering number* [4]. Informally, the δ-covering number $\mathcal{C}(\delta)$ of a set S is the minimum number of balls of radius δ needed to cover S. So it is a measure of the "volume" of S.

Theorem 1: For any $b_0 \in \mathcal{B}$, let $\mathcal{C}(\delta)$ be the δ-covering number of $\mathcal{R}(b_0)$. Given any constant $\epsilon > 0$, an approximation $V(b_0)$ of $V^*(b_0)$, with error $|V^*(b_0) - V(b_0)| \leq \epsilon$, can be found in time

$$O\left(\mathcal{C}\left(\frac{(1-\gamma)^2\epsilon}{4\gamma R_{\max}}\right)^2 \log_\gamma \frac{(1-\gamma)\epsilon}{2R_{\max}}\right).$$

However, for many realistic robotics tasks, the assumption of small $\mathcal{R}(b_0)$ may not hold. We would like our algorithm to do well when $\mathcal{R}(b_0)$ may be large, but $\mathcal{R}_{\pi^*}(b_0)$, the space reachable under an optimal policy π^*, is small. As $\mathcal{R}_{\pi^*}(b_0)$ is often much smaller than $\mathcal{R}(b_0)$, the assumption of small $\mathcal{R}_{\pi^*}(b_0)$ is more likely to hold. Unfortunately, this relaxed assumption is too weak, and the problem of computing approximate POMDP solutions remains hard, despite the assumption [4].

Theorem 2: Let b_0 be any point in $\mathcal{B}$ and π^* be an optimal policy. Given a constant $\epsilon > 0$, computing an approximation $V(b_0)$ of $V^*(b_0)$, with error $|V(b_0) - V^*(b_0)| \leq \epsilon |V^*(b_0)|$, is NP-hard, even if the covering number of $\mathcal{R}_{\pi^*}(b_0)$ is polynomial-sized.

On the other hand, if we are given a set of balls of radius δ that covers $\mathcal{R}_{\pi^*}(b_0)$, the problem becomes much easier [4]. We call the set C, which contains the centers of this set of balls, a *δ-cover* of $\mathcal{R}_{\pi^*}(b_0)$.

Theorem 3: For any $b_0 \in \mathcal{B}$ and any optimal policy π^*, given a proper δ-cover C of $\mathcal{R}_{\pi^*}(b_0)$ with $\delta = \frac{(1-\gamma)^2\epsilon}{2\gamma R_{\max}}$, an approximation $V(b_0)$ of $V^*(b_0)$, with error $|V^*(b_0) - V(b_0)| \leq \epsilon$, can be found in time

$$O\left(|C|^2 + |C| \log_\gamma \frac{(1-\gamma)\epsilon}{2R_{\max}}\right),$$

where $|C|$ is the size of C and $R_{\max} = \max_{s,a} |R(s,a)|$ is the maximum one-step reward.

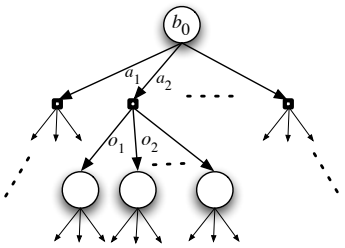


Fig. 2. The belief tree $T_{\mathcal{R}}$ rooted at b_0.

Together, Theorems 2 and 3 say that computing approximate POMDP solutions is hard, but the problem becomes much easier, if a proper δ-cover of $\mathcal{R}_{\pi^*}(b_0)$ is given. It follows that the key difficulty must lie in computing such a cover. Once the cover is obtained, we can find an approximate POMDP solution in time polynomial in the cover size. So, instead of following the common approaches of directly approximating V^* or searching for π^*, our SARSOP algorithm focuses on finding an approximate cover of $\mathcal{R}_{\pi^*}(b_0)$ through sampling.

Since there may be multiple optimal policies, SARSOP aims to sample $\mathcal{R}^*(b_0) = \bigcup_{\pi^*} \mathcal{R}_{\pi^*}(b_0)$, the union of all optimally reachable spaces.

In the following, to simplify the notations, we omit the argument b_0 in $\mathcal{R}(b_0)$ and $\mathcal{R}^*(b_0)$. It is understood that $\mathcal{R}$ and $\mathcal{R}^*$ are reachable from a given initial point b_0.

B. Overview of the Algorithm

SARSOP iterates over three main functions, SAMPLE, BACKUP, and PRUNE. A sketch is shown in Algorithm 1.

Like all point-based algorithm, SARSOP samples a set of points from the belief space. The sampled points form a tree $T_{\mathcal{R}}$ (Fig. 2). Each node of $T_{\mathcal{R}}$ represents a sampled point. As there is no confusion, we use the same symbol b 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 b_0. To sample a new point b', we pick a node b from $T_{\mathcal{R}}$ as well as an action $a \in \mathcal{A}$ and an observation $o \in \mathcal{O}$ according to suitable probability distributions or heuristics. We then compute b' using the formula

$$b'(s') = \tau(b,a,o) = \eta Z(s',a,o) \sum_s T(s,a,s') b(s),$$

where η is a normalization constant, and insert b' into $T_{\mathcal{R}}$ as a child of b. Clearly, every point sampled this way is reachable from b_0. If we apply all possible sequences of actions and observations, the set of nodes in $T_{\mathcal{R}}$ is exactly $\mathcal{R}$. The key is, of course, to avoid doing so and focus the sampling, instead, on $\mathcal{R}^*$.

To achieve this, SARSOP maintains both a lower bound $\underline{V}$ and an upper bound $\overline{V}$ on the optimal value function V^*. The set Γ of α-vectors represents a piecewise-linear approximation to V^* (Section II-A), and is also a lower bound when suitably initialized, using, *e.g.*, a fixed-action policy [1]. For the upper bound $\overline{V}$, SARSOP uses the sawtooth approximation [1]. The upper bound can be initialized in various ways, using the MDP or the Fast Informed Bound technique [1]. SARSOP uses the

Algorithm 2 Perform α-vector backup at a node b of $T_{\mathcal{R}}$.

BACKUP($T_{\mathcal{R}}$, Γ, b)

1: For all $a \in \mathcal{A}$, $o \in \mathcal{O}$, $\alpha_{a,o} \leftarrow \text{argmax}_{\alpha \in \Gamma}(\alpha \cdot \tau(b, a, o))$.
2: For all $a \in \mathcal{A}, s \in \mathcal{S}$,
$\alpha_a(s) \leftarrow R(s, a) + \gamma \sum_{o,s'} T(s, a, s')Z(s', a, o)\alpha_{a,o}(s')$.
3: $\alpha' \leftarrow \text{argmax}_{a \in \mathcal{A}}(\alpha_a \cdot b)$
4: Insert α' into Γ.

upper and the lower bounds to bias sampling towards $\mathcal{R}^*$ ((see Section III-C).

Next, we perform backup at selected nodes in $T_{\mathcal{R}}$. A backup operation at a node b collates the information in the children of b and propagates it back to b. We perform the standard α-vector backup (Algorithm 2), with the value function approximation represented as a set Γ of α-vectors. The value function approximation at b, obtained from the α-vector backup, is the same as that from the Bellman backup. However, the Bellman backup propagates only the value, while the α-vector backup propagates the gradient of the value function approximation along with the value to obtain a global approximation over the entire belief space rather than a local approximation at b.

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 (see Section III-D).

SARSOP is an anytime algorithm that returns the best policy found within a pre-specified amount of time. It gradually reduces the gap ϵ between the upper and lower bounds on the value function at b_0, until it reaches either a pre-specified gap size or the time limit.

C. Sampling

The NP-hardness result described in Section III-A suggests that sampling from $\mathcal{R}^*$ is hard. We use heuristics and information gathered from earlier samples to guide the sampling and improve the sampling distribution over time. Furthermore, by using value function bounds, we try to avoid sampling in regions that are unlikely to be reachable under any optimal policy, *i.e.*, outside of $\mathcal{R}^*$. See Algorithm 3 for the pseudocode.

To sample new belief points, SARSOP sets a target gap size ϵ between the upper and lower bound at the root b_0 of $T_{\mathcal{R}}$ and traverses a single path down $T_{\mathcal{R}}$ by choosing at each node the action with the highest upper bound and the observation that makes the largest contribution to the gap at the root of $T_{\mathcal{R}}$. This is the same action and observation selection strategy used in HSVI2 [19]. The sampling path is terminated under suitable conditions. Together, the strategies for action and observation selection and the choice of termination conditions control the resulting sampling distribution.

One termination condition is to stop when the sampling path reaches a node whose gap between the upper and lower bounds is smaller than $\gamma^{-t}\epsilon$, where t is the depth of the node in $T_{\mathcal{R}}$ [19]. If each leaf of $T_{\mathcal{R}}$ has a gap smaller than $\gamma^{-t}\epsilon$, the gap at the root is guaranteed to be smaller than ϵ. This condition, although reasonable, is inadequate. As the target gap ϵ at the root gets smaller, the sampling path must traverse deeper down the tree. As we go down the tree, the set of points in $\mathcal{R}$ increases much faster than the set of points in $\mathcal{R}^*$, and it becomes increasingly difficult to sample from $\mathcal{R}^*$. To focus sampling near $\mathcal{R}^*$ and minimize sampling in $\mathcal{R} \backslash \mathcal{R}^*$, we would like to make the sampling path as shallow as possible while still achieving the target gap ϵ at the root of $T_{\mathcal{R}}$. A potential dilemma here is that some nodes with high expected rewards lie deep in the tree, and we must allow the sampling path to go deep enough in order to reach them.

a) Selective deep sampling: As each backup operation chooses the action that *maximizes* the expected reward, improvements in lower bounds are quickly propagated to the root when nodes with high expected rewards are found. This not only directly improves the policy but also provides information to stop sampling more quickly in regions that are likely outside $\mathcal{R}^*$. In contrast, upper bounds cannot be propagated beyond a node until the upper bounds for *all* the actions at the node are sufficiently improved. Finding the best action is not enough. Thus we give preference to lower bound improvements and continue down a sampling path beyond the node with a gap of $\gamma^{-t}\epsilon$, if we predict that doing so likely leads to improvement in the lower bound at the root.

To make such a predication, conceptually we predict the optimum value $V^*(b)$ at a node b and propagate the predicted value $\hat{V}$ up towards the root. If $\hat{V}$ improves the lower bound at the root, we expand b and then repeat the procedure at the next selected node down the sampling path. Otherwise, we proceed to check the gap termination criterion described in the next subsection.

To predict the optimal value $V^*(b)$, we use a simple learning technique. We cluster beliefs according to suitable features and use previously computed values of beliefs in the same cluster as b to predict the value of b. This allows us to learn which parts of the belief space is worth exploring. Currently, we use the initial upper bound and the entropy of b as the features and discretize the belief space into a finite number of bins according to these two features. The average value of beliefs in a bin is used as the prediction for the value of any new belief falling into the bin. If a bin is empty, the initial upper bound of the new belief is used as its predicted value.

To implement this idea efficiently, we do not actually propagate the predicted value $\hat{V}$ back to the root. Instead, we pass a lower-bound target level L down the sampling path. The predicted value $\hat{V}$ is checked against L. If $\hat{V}$ fails to meet the target L at a node b, the lower bound at b will not be propagated further up towards the root of $T_{\mathcal{R}}$.

Let us now consider how to pass the target L at a node b to a child node $b' = \tau(b, a, o)$. First, observe that value function information is propagated up from b' to b only if the action a that takes b to b' has higher value than all other actions at b. We thus calculate an intermediate target level L' for a, which is set to the maximum over L and the values of all the actions at b (Algorithm 3, lines 7–8). Next, observe that the lower bound on the value of action a is

$$\underline{Q}(b, a) = \sum_s R(s, a)b(s) + \gamma \sum_o p(o|b, a)\underline{V}(b').$$

Algorithm 3 Sampling near $\mathcal{R}^*$.

SAMPLE($T_\mathcal{R}$, Γ)

1: Set L to the current lower bound on the value function at the root b_0 of $T_\mathcal{R}$. Set U to $L + \epsilon$, where ϵ is the current target gap size at b_0.
2: SAMPLEPOINTS($T_\mathcal{R}$, Γ, b_0, L, U, ϵ, 1).

SAMPLEPOINTS($T_\mathcal{R}$, Γ, b, L, U, ϵ, t).

3: Let $\hat{V}$ be the predicted value of $V^*(b)$.
4: **if** $\hat{V} \leq L$ and $\overline{V}(b) \leq \max\{U, \underline{V}(b) + \epsilon\gamma^{-t}\}$ **then**
5: **return**
6: **else**
7: $\underline{Q} \leftarrow \max_a \underline{Q}(b, a)$.
8: $L' \leftarrow \max\{L, \underline{Q}\}$.
9: $U' \leftarrow \max\{U, \underline{Q} + \gamma^{-t}\epsilon\}$.
10: $a' \leftarrow \arg\max_a \overline{Q}(b, a)$.
11: $o' \leftarrow \arg\max_o p(o|b, a') \left(\overline{V}(\tau(b, a', o)) - \underline{V}(\tau(b, a', o))\right)$.
12: Calculate L_t so that $L' = \sum_s R(s, a')b(s) + \gamma\left(p(o'|b, a')L_\mathrm{t} + \sum_{o \neq o'} p(o|b, a')\underline{V}(\tau(b, a', o))\right)$.
13: Calculate U_t so that $U' = \sum_s R(s, a')b(s) + \gamma\left(p(o'|b, a')U_\mathrm{t} + \sum_{o \neq o'} p(o|b, a')\overline{V}(\tau(b, a', o))\right)$.
14: $b' \leftarrow \tau(b, a', o')$.
15: Insert b' into $T_\mathcal{R}$ as a child of b.
16: SAMPLEPOINTS($T_\mathcal{R}$, Γ, b', L_t, U_t, ϵ, $t + 1$).

Hence the target level for b' is the value needed for $\underline{Q}(b, a)$ to achieve its target L' (Algorithm 3, line 12).

To guard against misleading predictions that result in unnecessarily deep samplings paths, we only continue down a sampling path until the gap between the upper and lower bounds is $\kappa\gamma^{-t}\epsilon$ for some $\kappa < 1$. The κ value is set to 0.5 in our current implementation.

b) The gap termination criterion: If our prediction shows no improvement of the lower bound at the root, we use the target gap size ϵ at the root to decide whether to terminate the sampling path and avoid sampling in regions unlikely to be in $\mathcal{R}^*$. As mentioned earlier, the straightforward way of achieving the target gap size ϵ between the upper and lower bounds at the root of $T_\mathcal{R}$ is to require a gap size $\gamma^{-t}\epsilon$ for all leaves of $T_\mathcal{R}$. However, it is in fact sufficient to ensure that the condition is satisfied somewhere along all the paths from the root to the leaves, rather than at the leaves themselves. This has the advantage of leveraging information globally from the other parts of $T_\mathcal{R}$ to terminate a sampling path as early as possible and thus improving computational efficiency.

To do this, we pass an upper-bound target level U down the sampling path as well. For a node b at depth t, we can terminate sampling if its upper bound is lower than $\underline{V}(b) + \epsilon\gamma^{-t}$ or the upper-bound target U passed down from parent of b. This termination criterion has the same effect as requiring all leaves to have a gap of no more than $\epsilon\gamma^{-t}$: if all leaves in $T_\mathcal{R}$ meet this termination criterion, the root b_0 achieves the target gap size of ϵ. The upper-bound target level U can be passed down a sampling path in a way similar to that for the lower-bound target level L. See Algorithm 3, lines 9 and 13.

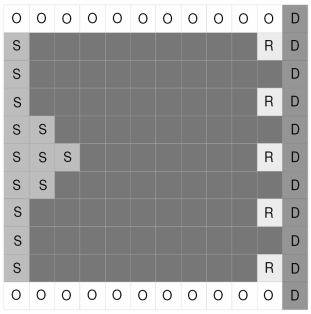
The combination of selective deep sampling and the gap termination criterion leads to an effective sampling strategy that goes deep into $T_\mathcal{R}$ when need. This avoids unnecessarily sampling in $\mathcal{R}\backslash\mathcal{R}^*$ and gives a better approximation to $\mathcal{R}^*$.

D. Pruning

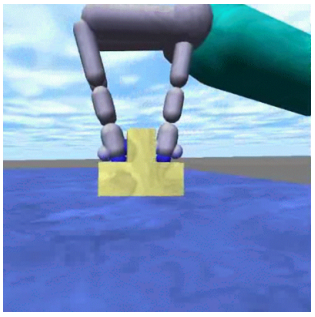
The efficiency of backup operations, which take up a significant fraction of the total computation time, depends significantly on the size of the set Γ of α-vectors. To improve computational efficiency, existing point-based algorithms usually prune an α-vector from Γ if it is dominated by others over the entire belief space $\mathcal{B}$. The notion of optimally reachable space suggests an alternative and more aggressive pruning technique: ideally, we want to prune an α-vector if it is dominated by other α-vectors over $\mathcal{R}^*$, rather than $\mathcal{B}$. Since $\mathcal{R}^*$ is potentially much smaller than $\mathcal{B}$, this may substantially reduce the size of Γ and improve the efficiency of the backup operations and thus the overall algorithm.

As $\mathcal{R}^*$ is not known in advance, we use B, the set of all sampled belief points contained in $T_\mathcal{R}$, as an approximation. To improve this approximation and to keep the size of B small, we prune from B those points that are provably suboptimal and do not lie in $\mathcal{R}^*$. For a node b in $T_\mathcal{R}$, if $\overline{Q}(b, a) < \underline{Q}(b, a')$ for two actions a and a', then we prune all the sampled points in the subtree resulting from taking action a at b, as an optimal policy will never take the action a at b and traverse the subtree underneath. It is possible that some pruned points may turn out to lie in $\mathcal{R}^*$, as there are other paths in $T_\mathcal{R}$ to reach them under an optimal policy. However, the benefits of keeping B small usually outweighs the loss in the approximation quality due to over-pruning. These points can also be eventually recovered from the other paths in $T_\mathcal{R}$.

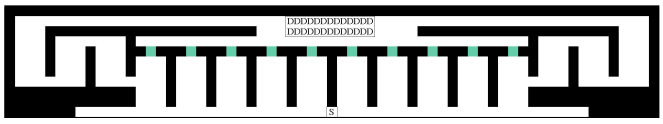
Belief point pruning in turn enables more aggressive α-vector pruning. In SARSOP, an α-vector is pruned if it is dominated by others over B. A simple criterion for dominance is to say that for two α-vectors α_1 and α_2, α_1 dominates α_2 at a belief point b if $\alpha_1 \cdot b \geq \alpha_2 \cdot b$. However, this is not robust. The set B is a finitely sampled approximation of $\mathcal{R}^*$. Since SARSOP computes an approximately optimal policy over B only, the computed policy may choose an action that causes it to slightly veer off $\mathcal{R}^*$ and get into a region in which the value function approximation is poor. To address this issue, we impose the more stringent requirement of dominance over a δ-neighborhood: α_1 dominates α_2 at a belief point b if $\alpha_1 \cdot b' \geq \alpha_2 \cdot b'$ at every point b' whose distance to b is less than δ, for some fixed constant δ. We call this *δ-dominance*. We can check δ-dominance very quickly by computing the distance d from b to the intersection of the hyperplanes represented by α_1 and α_2 and making sure that $d \geq \delta$. In the implementation, the value of δ can be set adaptively according to the effectiveness of α-vector pruning. A similar idea for α-vector pruning, but without using the δ-neighborhood, is described in [15].



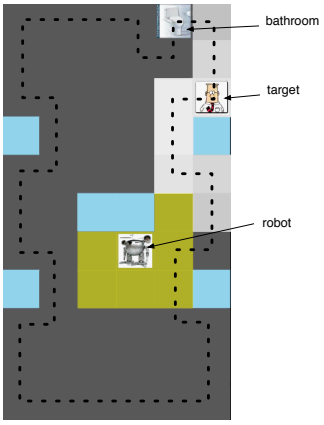
(*a*) Underwater Navigation, an instance of coastal navigation, shown on a reduced map with a 11×12 grid. "S" marks the possible initial positions for the robot. The robot is equally likely to start in any of these positions. "D" marks the destinations. "R" marks the rocks. "O" marks places that the robot can fully localize itself.



(*b*) Grasping. A fingered robot arm grasps a stepped block. Courtesy of L.P. Kaelbling and T. Lozano-Pérez.



(*c*) Integrated Exploration. A robot navigates with an uncertain map. Areas shaded in black represent obstacles. Areas shaded in light gray represent (possibly damaged) bridges. "S" marks the start location for the robot. "D" marks destination locations.



(*d*) Homecare. A robot follows a moving person, the target. The light blue areas indicate obstacles. The black dashed curve indicates the target's path. The green area around the robot indicates the the robot sensor' visibility region. The various shades of gray show the robot's belief of the current target position.

Fig. 3. Some common robotic tasks modeled as POMDPs.

IV. Experiments

We have successfully applied SARSOP to a set of distinct robotic tasks. In this section, we describe these tasks, the experimental setup, and the results.

A. Robotic Tasks Studied

Uncertainty arises in various ways in robotic systems. Suppose that the state of a robotic system is given by $(x_{\mathrm{r}}, x_{\mathrm{e}})$, where x_{r} represents the state of the robot and x_{e} represents the state of the environment. Inaccuracies in robot control and sensing are the typical causes for uncertainty in x_{r}. They are almost always present to some degree. Uncertainty in x_{e}, On the other hand, varies widely. We thus divide the robotic tasks studied here into three categories according to the uncertainty in x_{e}. In the first category, the environment is static and known with high accuracy. So uncertainty in x_{e} can be ignored, and we only need to consider uncertainty in x_{r} in planning the robot's actions. In the second category, the environment is static, but not known accurately. Thus, we must take into account the uncertainty in both x_{r} and x_{e} in planning. In the last category, the environment is not static and changes over time. We need a dynamic model of the environment and use it to plan actions for the robot to respond to changes in the environment.

a) Underwater Navigation: We start with an instance of the well known coastal navigation problem. An autonomous underwater vehicle (AUV) navigates in an environment modeled as a 51×52 grid map (Fig. 3*a*). The AUV needs to navigate from the left border of the map to the right border. It must avoid rocks scattered near the goals, as they may cause severe damages to the vehicle. In each step, the AUV can either stay in the current position or move to any of the four adjacent positions (directly above, below, left, and right). Due to poor visibility conditions, the AUV can only localize itself along the top or bottom borders, where there are beacon signals. The environment is static and known in advance. So this problem belongs to the first category.

Roughly, the optimal policy for the AUV is to move diagonally until it reaches the top or bottom border to localize itself. It can then safely pass through the rocks and get to the destinations on the right border. A feature of this problem is that heuristics assuming full observability (*e.g.*, an MDP policy) favor shorter horizontal paths rather than diagonal paths and thus often choose the wrong action.

b) Grasping: This problem was introduced in the work of Hsiao, Kaelbling, and Lozano-Pérez [3]. As a POMDP, this problem is similar to coastal navigation: the environment is static and known, but due to limited sensing capabilities, the robot has difficulty in determining its own state exactly. It needs to perform information-gathering actions to reduce the state uncertainty in order to reach the goal. However, as a robotic task, grasping has quite different physical characteristics. Here, a two-dimensional Cartesian robot arm with two fingers tries to grasp a stepped block on a table (Fig. 3*b*). It has only contact sensors at the tip and the sides of each finger to help determine the state. The robot performs compliant guarded moves (left, right, up, and down) and always maintains contact with the surface of the block or the boundary of the environment at the beginning and end of each move. The goal is to move the robot arm and have its two fingers straddle the block so that grasping is possible. More details on this problem can be found in [3].

c) Integrated Exploration: For some tasks, robots must traverse an area whose map is highly uncertain, for example, when robots perform SLAM tasks. In this situation, the robot must gather information to reduce map uncertainty, localize itself, and navigate to reach the goal. This is sometimes

called *integrated exploration* [8]. When the environment is static, integrated exploration belongs to our second category. Unfortunately, despite a static environment, uncertainty in the environment map causes the number of states for x_{e} to grow exponentially. Recall further that increase in the number of states in turn causes the belief space size to grow exponentially. Currently, such doubly exponential growth is too difficult to manage, even for point-based POMDP algorithms.

Our problem here models a similar, but simplified scenario (Fig. 3*c*). In one step, the robot can move from its current location to one of the eight adjacent locations horizontally, vertically, and diagonally. The result of a move is uncertain. The robot can localize itself in several locations scattered around the environment. To reach the destination, the robot may follow one of the long routes along the far left and right sides of the environment or take a shortcut through one of the bridges (shaded in light gray in Fig. 3*c*). Due to flood damages, at most two bridges are still functional. The robot's goal is to reach the destination nodes as quickly as possible, using such an uncertain environment map. Even in this simplified setting, we still end up with more than 15,000 states.

d) Rock Sample: The Rock Sample problem first appeared in the work on HSVI [18]. In this problem, a rover explores an area modeled as a small grid and looks 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 sensor 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.

In this problem, the environment is static, and a map with exact rock positions is available. However, the environment map that really matters is the one that marks the positions of *valuable* rocks. This map is unknown in advance. and the rover must infer this map from sensor readings. So this problem can be regarded as an instance of integrated exploration.

e) Tag: The Tag problem first appeared in the work on PBVI [10]. In Tag, the robot's goal is to follow a target that intentionally moves away. The robot and the target operate in a grid environment with 29 positions in total. In one step, they can either stay or move one of four adjacent positions (above, below, left, and right). The robot always knows its own position, but can observe the target's position only if they are in the same position. The robot pays a cost for each move and receives a reward every time it arrives in the same position as that of the target. Here, the environment changes over time due to the target motion. Thus the problem belongs to the third category.

f) Homecare: This problem models a robot following a person around at home for caretaking purposes (Fig. 3*d*). It is related to Tag, but involves a much larger number of states and more complex environment dynamics. Imagine that an elderly person moves around at home. His motion is non-deterministic: he follows a fixed path (marked as a black dashed curve in Fig. 3*d*), but in each time step, he may pause or proceed along the path with equal probabilities. Along the path, there is special location representing a bathroom, where the person may stay for an extended duration. The person has a call button to call the robot over for help. The call button stays on for some uncertain duration and then goes off. The robot gets a reward only if it arrives in time. The robot can observe the person's position when they are close enough. Clearly the robot should stay close to the person in order to track his position well and improve the chance of receiving rewards. At the same time, it also wants to minimize movement in order to reduce power consumption. POMDP provides a principled way to evaluate such trade-offs.

B. Experimental Setup

We applied SARSOP to the above 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 SARSOP for a maximum of two hours to reach this level and recorded the resulting policy. To estimate the expected total reward of the policy, we performed sufficiently large number of simulation runs until the variance in the estimated value was small. For comparison, we also ran HSVI2 on these tasks, following the same procedure. Both algorithms are implemented in C++. They were compiled with g++ v4.1.2. The experiments were performed on a PC with a 2.66GHz Intel processor and 2GB memory. For HSVI2, we used the newest software released by its original author, zmdp v1.1.3, which is a highly optimized implementation.

For SARSOP, the δ value for α-vector pruning was set at 1×10^{-2} for the two largest problems, Rock Sample and Homecare, and 1×10^{-4} for the rest. The performance of SARSOP is affected by the δ value, but not sensitive to it. One important consideration in the choice of δ is the dimensionality of the belief space involved, *i.e.*, the number of states. The rough guide that we have been using is 1×10^{-2} for POMDPs with about 10,000 states or more and 1×10^{-4} for those with substantially fewer states. We are currently implementing an adaptive technique to set δ automatically and will include it in the final software release.

C. Results

The results are shown in Table I. Column 2 of the table lists the estimated expected total rewards for the computed policies and the 95% confidence intervals. Column 3 lists the corresponding computation times.

For all six tasks, SARSOP obtained good approximate solutions within the two-hour limit. In five out of the six tasks, SARSOP substantially outperformed HSVI2, sometimes by several times. For two tasks (Integrated Exploration and Tag), HSVI2 was unable to reach a comparable reward level as that of SARSOP within the two-hour time limit. Thus, for these two tasks, we also report the reward level that HSVI2 was able to reach at the end of two hours (Table I).

TABLE I

PERFORMANCE COMPARISON.

	Reward	Time (s)
Underwater Navigation,		
$\|\mathcal{S}\|$=2,653,$\|\mathcal{A}\|$=6,$\|\mathcal{O}\|$=103		
SARSOP	722.59 ± 1.30	72
HSVI2	721.45 ± 0.75	720
Grasping		
$\|\mathcal{S}\|$=1,253,$\|\mathcal{A}\|$=6,$\|\mathcal{O}\|$=96		
SARSOP	320.00 ± 0.16	8
HSVI2	319.88 ± 0.14	60
Integrated Exploration		
$\|\mathcal{S}\|$=15,517,$\|\mathcal{A}\|$=8,$\|\mathcal{O}\|$=1,015		
SARSOP	$(1.58 \pm 0.03) \times 10^6$	5,400
HSVI2	$(1.41 \pm 0.02) \times 10^6$	5,400
	$(1.43 \pm 0.02) \times 10^6$	7,200
Rock Sample (7,8)		
$\|\mathcal{S}\|$=12,545,$\|\mathcal{A}\|$=13,$\|\mathcal{O}\|$=2		
SARSOP	21.27 ± 0.13	400
HSVI2	21.27 ± 0.09	250
Tag		
$\|\mathcal{S}\|$=870,$\|\mathcal{A}\|$=5,$\|\mathcal{O}\|$=30		
SARSOP	-6.13 ± 0.12	6
HSVI2	-7.43 ± 0.11	6
	-6.40 ± 0.10	7,200
Homecare		
$\|\mathcal{S}\|$=5,408,$\|\mathcal{A}\|$=9,$\|\mathcal{O}\|$=928		
SARSOP	16.86 ± 0.45	960
HSVI2	16.88 ± 0.37	2,880

On Rock Sample, SARSOP did not perform as well as HSVI2 for a very specific reason. HSVI2 implements an α-vector masking technique, which opportunistically computes only selected entries in the α-vectors. This technique is particularly beneficial here, because in Rock Sample, the robot position is fully observed, which substantially reduces the overall level of uncertainty involved. Furthermore, the remaining state variables that specify the status of rocks are independent, which also helps to improve the effectiveness of masking. Without masking, HSVI2 was only able to reach the reward level of 18.98 ± 0.09 after 400 seconds of computation time. This is worse than that of SARSOP. The effectiveness of masking degenerates for uncertain robot movements and noisy observations, which are the more common case in practice. For this reason, we currently do not incorporate masking in our implementation.

V. CONCLUSION

Point-based algorithms have greatly improved the speed of POMDP solution by sampling from the reachable space. This paper presents a new point-based algorithm, SARSOP, which exploits the notion of optimally reachable spaces to further improve computational efficiency. We applied SARSOP to a set of distinct robotic tasks, all modeled as POMDPs with a large number of states. SARSOP computed good approximate solutions to all of them in reasonable time. Further, it outperformed one of the fastest existing point-based algorithm in most of these tasks. These results indicate that approximating optimally reachable spaces through sampling is an interesting new angle to look at the problem. It has led to the more effective sampling and pruning strategies in SARSOP.

Along with other reports in literature [2, 3, 10, 11, 14, 19], our results indicate that with the advances in POMDP solution algorithms, the POMDP approach is gradually becoming practical for non-trivial robotic tasks. We have implemented SARSOP as a software package, and it is available for download at `http://motion.comp.nus.edu.sg/projects/pomdp/pomdp.html`.

Acknowledgements. We thank Yanzhu Du and Xan Huang for helping with the software implementation. This work is supported in part by AcRF grant R-252-000-327-112 from the Ministry of Education of Singapore.

REFERENCES

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

[2] 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.

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

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

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

[6] L. Kaelbling, M. Littman, and A. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1–2, pp. 99–134, 1998.

[7] O. Madani, S. Hanks, and A. Condon, "On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems," in *Proc. Nat. Conf. on Artificial Intelligence*, 1999, pp. 541–548.

[8] A. Makarenko, S. Williams, F. Bourgault, and H. Durrant-Whyte, "An experiment in integrated exploration," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, 2002.

[9] C. Papadimitriou and J. Tsisiklis, "The complexity of Markov decision processes," *Mathematics of Operations Research*, vol. 12, no. 3, pp. 441–450, 1987.

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

[11] J. Pineau, M. Montemerlo, M. Pollack, N. Roy, and S. Thrun, "Towards robotic assistants in nursing homes: Challenges and results," *Robotics & Autonomous Systems*, vol. 42, no. 3–4, pp. 271–281, 2003.

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

[13] M. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.

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

[15] G. Shani, R. Brafman, and S. Shimony, "Adaptation for changing stochastic environments through online POMDP policy learning," in *Proc. Eur. Conf. on Machine Learning*, 2005, pp. 61–70.

[16] ——, "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*, vol. 21, pp. 1071–1088, 1973.

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

[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.