2008 IEEE International Conference on
Robotics and Automation
Pasadena, CA, USA, May 19-23, 2008

A Point-Based POMDP Planner for Target Tracking

David Hsu* Wee Sun Lee* Nan Rong†

*Department of Computer Science
National University of Singapore
Singapore 117590, Singapore

†Department of Computer Science
Cornell University
Ithaca, NY 14853, USA

***Abstract*— Target tracking has two variants that are often studied independently with different approaches: *target searching* requires a robot to find a target initially not visible, and *target following* requires a robot to maintain visibility on a target initially visible. In this work, we use a partially observable Markov decision process (POMDP) to build a single model that unifies target searching and target following. The POMDP solution exhibits interesting tracking behaviors, such as anticipatory moves that exploit target dynamics, information-gathering moves that reduce target position uncertainty, and energy-conserving actions that allow the target to get out of sight, but do not compromise long-term tracking performance. To overcome the high computational complexity of solving POMDPs, we have developed SARSOP, a new point-based POMDP algorithm based on successively approximating the space reachable under optimal policies. Experimental results show that SARSOP is competitive with the fastest existing point-based algorithm on many standard test problems and faster by many times on some.**

I. Introduction

Target tracking is an important task for autonomous robots and has attracted considerable attention in recent years [1], [3], [4], [5], [12], [13], [14]. In the target tracking problem, we construct motion strategies for a robot equipped with visual sensors so that it can effectively track a moving target, despite obstruction by obstacles. More precisely, the robot should maneuver to keep the target visible to the mounted sensors. Target tracking finds many applications in security and surveillance, wildlife monitoring, and homecare settings. In particular, our work is part of a larger effort to build an autonomous homecare robot that watches over children or elderly people and recognizes their activities.

Target tracking has two main variants: *target searching* requires a robot to find a target initially not visible, and *target following*[1] requires a robot to maintain visibility on a target initially visible for the longest duration possible. The two variants are tackled with very different approaches in the literature. In this work, we show that by modeling target tracking as a partially observable Markov decision process (POMDP) [18], searching and following can be unified. The main idea is to represent the target position as a probability distribution, whether the target is visible to the robot sensors or not. So the target position is always "known" to the robot with some degree of uncertainty. The robot then chooses its actions according to a probabilistic model of target behaviors and a reward function that encourages the robot to keep the target visible.

The POMDP framework offers several other advantages. It provides a principled way to deal with uncertainties in robot control and sensing. It also easily incorporates additional requirements, *e.g.*, minimizing the robot's power consumption.

One main criticism of the POMDP framework is its high computational complexity. Solving POMDPs exactly is computationally intractable [15]. Intuitively, the intractability is due to the "curse of dimensionality": in a discrete POMDP, the belief space $\mathcal{B}$ used in solving a POMDP has dimensionality equal to $|S|$, the number of states in the POMDP. Thus the size of $\mathcal{B}$ grows exponentially with $|S|$. In a tracking problem, if both the robot and the target can occupy any of the 100 positions in an environment, the resulting belief space is *10,000-dimensional*! The high computational cost is a major barrier in applying the POMDP approach to solve realistic robot tasks.

However, worst-case analysis may not accurately reflect the difficulty of problems typically found in practice. Many interesting robot tasks can be solved in a potentially much smaller subset $\mathcal{R}(b_0) \subseteq \mathcal{B}$ [7], [8], [16], which contains only belief points reachable from a given initial belief point $b_0 \in \mathcal{B}$ under arbitrary sequences of actions. Indeed, we have proven recently that an approximately optimal POMDP solution can be computed efficiently, if $\mathcal{R}(b_0)$ is small in the sense that it can be "covered" by a small number of points [10]. Point-based POMDP algorithms, which compute approximate POMDP solutions over a set of *sampled points* in $\mathcal{R}(b_0)$, benefit significantly from this property. Our results in [10] also suggest a solution strategy when $\mathcal{R}(b_0)$ is not small: sample heuristically near $\mathcal{R}^*(b_0)$, the subset of belief points reachable from b_0 under *optimal* sequences of actions.

Using these theoretical insights, we have developed a new point-based POMDP algorithm called SARSOP, which stands for Successive Approximations of the Reachable Space under Optimal Policies. We have successfully applied SARSOP to tracking problems with more than 9,000 states and observed interesting tracking behaviors, *e.g.*, anticipatory moves that exploit target dynamics, information-gathering moves that reduce target position uncertainty, and energy-conserving actions that allow the target to get out of sight, but do not compromise tracking performance.

II. Related Work

There is a vast literature on target tracking, which includes both target searching and following. We can only selec-

[1]In the literature, this is sometimes called *target tracking*, but we use the more accurate term *target following* to differentiate it from the more general notion of tracking here.

tively touch on some work here. Target search is closely related the pursuit-evasion problem [21]. One interesting approach is to capture the visibility relationships through a cell decomposition of the environment and then search the decomposition graph for the target [3], [5]. Target following strategies [1], [4], [12], [13], [14] differ greatly, depending on the amount of prior information available. If both the environment and the target trajectory are completely known, optimal target following strategies can be computed through dynamic programming [12], though the high computational cost is high. SARSOP also uses a dynamic programming approach, but it is significantly more efficient by using only a set of sampled points from $\mathcal{B}$. If the environment and the target trajectory are both unknown in advance, one approach is to move the robot so as to minimize an objective function that tries to capture the short- and long-term risk of losing the target [1], [4], [13]. Other probabilistic approaches to target tracking include, *e.g.*, [22]

We use POMDP to build a single model that unifies searching and following. It assumes that the target position is always known up to some degree of uncertainty, modeled as a probability distribution, but the target may not be always visible. The POMDP framework also deals with uncertainties in robot control and sensing in a natural way.

One main difficulty with the POMDP approach is its high computational complexity. As a result, there has been a lot of work on computing approximate POMDP solutions. See [6] for a survey. In particular, point-based POMDP algorithms (*e.g.*, [7], [16], [19], [20]) have been very successful in computing good approximate solutions for large POMDPs with many states. They sample a set of points from the belief space $\mathcal{B}$ and use them as a compact representation of $\mathcal{B}$. Early point-based algorithms sample from the entire $\mathcal{B}$ using fixed- or variable-resolution grids. To improve computational efficiency, more recent POMDP algorithms sample only $\mathcal{R}(b_0)$. SARSOP follows this approach, but it further improves efficiency by focusing sampling on $\mathcal{R}^*(b_0)$, the subset of $\mathcal{B}$ most relevant to the POMDP solution.

Our POMDP tracking problem is related to the Tag problem described in [16]. However, the problems considered here involve a much larger number of states and more complex target behaviors. The SARSOP algorithm is also more efficient than the PBVI algorithm used in [16] and can handle more realistic target tracking tasks (see Section V).

Another potential difficulty with the POMDP approach is the acquisition of a good probabilistic model of target behavior, but machine learning techniques can help [2].

III. A POMDP Approach to Target Tracking

We start with a brief review of POMDPs. See [11] for a more complete introduction. We then describe how to model the target tracking problem as a POMDP.

A. Background on POMDPs

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

The agent always lies in some state $s \in S$. In each time step, it takes some action $a \in A$ and moves from a start state s to an end state 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 its current state. Due to the uncertainty in observation, the observation result $o \in O$ is again described as a conditional probability function $Z(s, a, o) = p(o|s, a)$ for $s \in S$ and $a \in A$.

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

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 *belief state*, or *belief*, for short. A belief is a probability distribution over S. A POMDP policy $\pi: \mathcal{B} \rightarrow A$ maps a belief $b \in \mathcal{B}$ to the prescribed action $a \in A$.

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 the optimal policy π^*, can be approximated arbitrarily closely by a convex and piecewise-linear function $V(b) = \max_{\alpha \in \Gamma}(\alpha \cdot b)$, where b is a discrete vector representation of a belief and Γ is a finite set of vectors called α-vectors. Each α-vector is associated with an action, and the policy can be executed by selecting the action corresponding to the best α-vector at the current belief b. So the policy can be represented by a set Γ of α-vectors. Policy computation, which, in this case, involves the construction of Γ, is usually performed offline.

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

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

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

B. Target Tracking as a POMDP

Our problem setting is motivated by homecare applications. Imagine that an elderly person moves around at home

and has a call button to call a robot over for help. The call status stays on for some time and then goes off. If the robot arrives while the call status is on, it gets a reward; otherwise, it gets no reward. Clearly the robot should stay close the person in order to improve the chance of receiving rewards, but at the same time, the robot needs to minimize movement in order to reduce power consumption. So the naive strategy of following right behind the person does not work well.

To formulate the problem as a POMDP, we model the environment as a regular grid. See Fig. 3 for examples. The robot and the target (in this case, the person with the call button) can occupy any of the grid cells that are free of obstacles. The state s of this POMDP is composed of the robot position x_{r}, the target position x_{t}, and the call status c: $s = (x_{\mathrm{r}}, x_{\mathrm{t}}, c)$. If the environment contains n free cells, then there are $n \cdot n \cdot 2 = 2n^2$ distinct states, resulting in a belief space of $2n^2$ dimensions.

In one time step, the target can stay where it is or move to a neighboring cell. The target motion is described by a given probability function T_{t}, conditioned on the target's current position: if the target is currently at x_{t}, it will be at x'_{t} in the next time step with probability $T_{\mathrm{t}}(x_{\mathrm{t}}, x'_{\mathrm{t}}) = p(x'_{\mathrm{t}}|x_{\mathrm{t}})$.

The person may turn on the call button in each step with probability p_1. If the call status is on, the person may turn it off with some probability p_2 in each time step, indicating that help is no longer needed. This model has two main implications. First, as the call duration follows the geometric distribution, the mean duration of a call is $1/p_2$, Second, most calls are short. The robot must arrive quickly in order to receive rewards, thus increasing the difficulty of tracking.

The robot motion resulting from an action is described similarly by another probability function T_{r}, conditioned on both the robot's current position and its action. The robot's actions consist of commands to stay where it is or to move to a neighboring cell. If the robot is currently at x_{r} and takes action a it will be at x'_{r} in the next time step with probability $T_{\mathrm{r}}(x_{\mathrm{r}}, a, x'_{\mathrm{r}}) = p(x'_{\mathrm{r}}|x_{\mathrm{r}}, a)$. Note that the robot may not be able to execute the commands perfectly due to control uncertainty. This can be modeled with a suitable T_{r}.

We assume that the robot can see the target through its sensors if they lie in the same or neighboring cells. Uncertainty on the target position due to sensor noise can be modeled in the observation probability function Z.

The robot receives a reward, if it reaches the cell that the target occupies while the call button is on. In one step, if the robot does not move, it incurs no costs (*i.e.*, negative rewards). Otherwise, it incurs a cost proportional to the distance traveled. The robot's goal is to maximize the expected total discounted reward.

The POMDP formulation does not explicitly differentiate whether the target is visible or not. To execute a policy, the robot maintains a belief of the target position. When the target is visible to the sensors and the sensor data are good, the belief is sharpened. When the target is not visible or the sensor data are poor, the belief becomes more diffuse. In the extreme case, when the target remains invisible for a long time, the belief may eventually converge to a uniform distribution. This way, target searching and target following are unified in a natural way. Clearly, if the robot knows the target position well, it can choose better actions and receive higher rewards. Therefore, an optimal policy favors sharp beliefs, while also taking into account the cost of obtaining them. In the next section, we show how approximately optimal policies can be computed efficiently.

IV. SARSOP

Most recent point-based POMDP algorithms sample from $\mathcal{R}(b_0)$, the set of points reachable from a given initial point $b_0 \in \mathcal{B}$, and maintain a set Γ of α-vectors. The set Γ represent a piecewise-linear lower-bound approximation $\underline{V}$ to the optimal value function V^* (see Section III-A). To improve the approximation $\underline{V}$, we perform backup operations on the α-vectors at the sampled points. A backup operation are essentially an iteration of dynamic programming, which improves the approximation by looking ahead one step further. With suitable initialization (using, *e.g.*, fixed-action policies [6]), $\underline{V}$ is always a lower bound on V^*, and converges to V^* under suitable conditions [10], [16], [19].

Our theoretical analysis in [10] provides two insights on point-based POMDP algorithms:

- POMDPs are easy to solve, when the reachable space $\mathcal{R}(b_0)$ is small, in the sense that it can be "covered" by a small number of points. Even if $\mathcal{R}(b_0)$ is large, $\mathcal{R}^*(b_0)$, the space reachable under an optimal policy, may be much smaller, as $\mathcal{R}^*(b_0) \subseteq \mathcal{R}(b_0)$. Trying to sample $\mathcal{R}^*(b_0)$ may reduce computation time. However, the difficulty is that neither the optimal policy nor $\mathcal{R}^*(b_0)$ is known in advance, and we must approximate $\mathcal{R}^*(b_0)$.
- A good approximation of V^*, and thus the optimal policy, can be achieved locally by nearest neighbor interpolation over a set of sampled points.

SARSOP embodies these insights in two corresponding ways:

- It computes successive approximations to $\mathcal{R}^*(b_0)$ through heuristic sampling and pruning away sampled points that are provably suboptimal.
- It prunes away α-vectors when they are dominated locally at the sampled points, which are an approximation to $\mathcal{R}^*(b_0)$ rather than over the entire belief space. This results in a much smaller set of α-vectors and reduced running time.

A complete description of SARSOP can be found in [9]. Below we give a self-contained summary. In the following, to simplify the notation, we will 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.

A. Overview of the Algorithm

A sketch of SARSOP is shown in Algorithm 1. The algorithm iterates over three main functions, SAMPLE, BACKUP, and PRUNE, until further update produces little change in the lower-bound approximation $\underline{V}$.

The sampled belief points form a tree X (Fig. 1). Each node of X represents a sampled point. As there is no

Algorithm 1 SARSOP.

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

SAMPLE(X, Γ)

1: Pick $b \in X$, $a \in A$, and $o \in O$.
2: $b' \leftarrow \tau(b, a, o)$.
3: Insert b' into X as a child of b.

BACKUP(X, Γ, b)

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

confusion, we use the same symbol b to denote both a sampled point and its corresponding node in X. The root of X is the initial belief point b_0. To sample a new point b', we pick a node b from X as well as an action $a \in A$ and an observation $o \in O$ according to suitable probability distributions or heuristics. We then compute $b' = \tau(b, a, o)$ and insert b' into X as the child of b. Clearly, every point sampled this way is reachable from b_0.

SARSOP currently uses a sampling strategy similar to that of HSVI2 [19]. To sample a new belief point, it traverses a single path down the tree X by choosing the action with the highest upper bound on the value function and the observation that maximally reduces the gap between the lower and the upper bounds at the root of X, until it reaches a node b at a desired level. It then samples a new point b' and insert b' into X as a child of b (see SAMPLE in Algorithm 1). After creating the new node, it performs backup operations at all nodes along the path leading to b'.

A backup operation at a node b collates the information in the children of b and propagates it back to b. We first find all beliefs reachable from b with a single action a and an observation o. For each of these beliefs $b' = \tau(b, a, o)$, we find, in the set Γ, the α-vector $\alpha_{a,o}$ having the largest inner product with b'. We then combine all such α-vectors $\alpha_{a,o}$ associated with a particular action a into a single vector α_a. Finally, we find, among the vectors α_a, the vector α' having the largest inner product with b and add α' to Γ. The value function approximation at b, obtained from this α-vector backup, is the same as that from the standard 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.

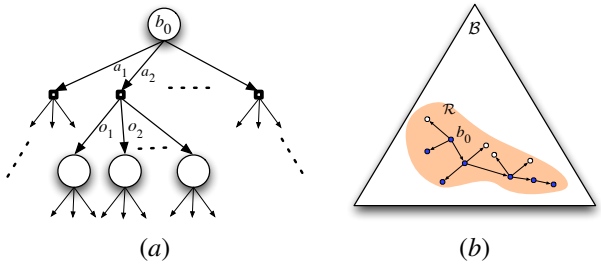


Fig. 1. (a) The POMDP tree with the root b_0. (b) The corresponding reachable belief space $\mathcal{R}$. The shaded nodes indicate points in $\mathcal{R}^*$.

B. The Reachable Space under Optimal Policies

The efficiency of point-based POMDP algorithms depends on two key factors. First, how are the sampled points distributed? Since our goal is to compute an optimal policy for a given initial belief point b_0, we should avoid sampling far away form $\mathcal{R}^*$ to reduce computational cost. Second, the cost of a single backup operation is directly proportional to the number of α-vectors in Γ. One way of speeding up backup, which dominates the total running time, is then to keep the set Γ small. Existing point-based algorithms usually prune an α-vector if it is dominated by others over the entire $\mathcal{B}$. We would like to prune more aggressively: an α-vector is pruned if it is dominated by other α-vectors over $\mathcal{R}^*$. This results in potentially much smaller Γ.

Unfortunately we do not know the optimal policies or the optimally reachable space $\mathcal{R}^*$ in advance. Even if we do, the cost of checking α-vector dominance over $\mathcal{R}^*$ exactly is prohibitive. Our theoretical analysis [10] suggests that checking for local dominance around the sampled points is often sufficient for good performance. Doing so greatly reduces the cost of pruning. The details are described in next two subsections.

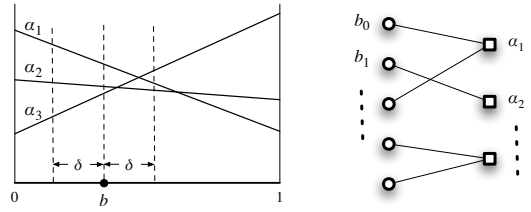
C. Approximating $\mathcal{R}^*$ through Belief Point Pruning

To use the sampled points in X as an approximation of $\mathcal{R}^*$, we must prune those points far away from $\mathcal{R}^*$. For this, we maintain not only a lower bound $\underline{V}$, but also an upper bound $\overline{V}$ on the optimal value function. Various methods for maintaining the upper bound can be used. Currently SARSOP uses the sawtooth approximation [6].

By maintaining both upper and lower bounds, we can improve our approximation of $\mathcal{R}^*$ as these bounds improve. Consider a node b in X. Let

$$\underline{Q}(b, a) = \sum_{s \in S} R(s, a) b(s) + \gamma \sum_{o \in O} p(o|b, a) \underline{V}(\tau(b, a, o))$$

be the lower bound on the value of taking action a at b. The upper bound $\overline{Q}$ is defined similarly, using $\overline{V}$. If $\overline{Q}(b, a) < \underline{Q}(b, a')$ for some actions a and a', the optimal policy will never take the action a at node b and traverse the subtree underneath. We thus prune this subtree along with all the associated sampled points. Some pruned points may actually lie in $\mathcal{R}^*$, as they are reachable from other paths in X under the optimal policy. However, the benefit of reducing the number of sampled points usually outweighs



(*a*) (*b*)

Fig. 2. (*a*) δ-dominance. α_1 dominates α_2, but not α_3 in the δ-neighborhood of b. (*b*) The certificate structure.

the loss in value function approximation due to over-pruning. These points can also be eventually recovered from the other paths in X, if they indeed lie in $\mathcal{R}^*$.

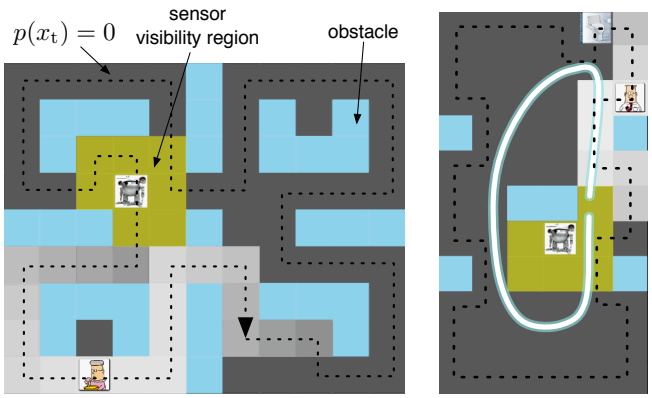
Belief point pruning improves computational efficiency in two ways. As the suboptimal branches of X are pruned, SARSOP avoids sampling the part of $\mathcal{B}$ unreachable under the optimal policies (see SAMPLE in Algorithm 1). Thus the sample distribution is automatically adapted to bias towards $\mathcal{R}^*$. Furthermore, belief point pruning helps α-vector pruning. We explain why in the next subsection.

D. α-Vector Pruning

Let P denote the set of sampled belief points in X. SARSOP prunes away an α-vector if it is dominated by others over $\mathcal{R}^*$, which is approximated by the current P. 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 P is a finitely sampled approximation of $\mathcal{R}^*$. Since SARSOP computes an approximately optimal policy over P 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 $d \geq \delta$. See Fig. 2*a* for an example. A similar idea for α-vector pruning, but without using the δ-neighborhood, is described in [17].

To prune α-vectors efficiently, SARSOP maintains a *certificate* structure (Fig. 2*b*). The certificate structure is a bipartite graph consisting of two sets of nodes, P and Γ. There is an edge between two nodes $b \in P$ and $\alpha \in \Gamma$, if α is not dominated by another α-vector over the δ-neighborhood of b. Thus, every edge (b, α) represents a certificate that demonstrates the usefulness of α. A sampled point b issues a certificate to an α-vector, when it is created through a backup operation at b. The certificates from each sampled point b are checked periodically and revoked if the corresponding α-vectors are δ-dominated at b. Following the checks, any α-vector holding no certificates can be immediately pruned.

By maintaining the certificate structure and checking δ-dominance, SARSOP prunes away the useless α-vectors, but retain all the optimal α-vectors over the space within a distance δ of $\mathcal{R}^*$.



(*a*) (*b*)

Fig. 3. Simulation experiments for target tracking.

The pruning method above works well if P is a representative sample of $\mathcal{R}^*$. At the beginning, when there are few sampled points in P, pruning the α-vectors too aggressively may result in poor performance. To mitigate this effect, we incorporate all the corner points of the belief simplex $\mathcal{B}$ into the certificate structure. These additional points are not used for backup operations, but only for checking δ-dominance. They introduce little overhead, as the sparsity of these points (with only one nonzero coefficient per point) allows us to compute the certificates involving them very quickly.

We expect each belief point to be involved in only a few certificates. Thus, as a number of belief points decreases, the number of certified α-vector decreases as well. This is why belief point pruning helps in α-vector pruning.

V. Experiments

A. Target Tracking Simulations

We used SARSOP to compute tracking policies in several simulated environments. See Fig. 3 for examples. The light blue areas in the figures indicate obstacles. The black dashed curve indicates the target's path. The target motion is non-deterministic: it follows this path, but in each time step, it may pause or proceed along the path with equal probabilities. The green area around the robot indicates the robot sensor' visibility region. The various shades of gray show the robot's belief of the target position. Lighter color indicates higher probability. To focus on target tracking behaviors, we assume in these experiments that there is no uncertainty in robot control and sensing. The robot can execute motion commands and observe its own position and call status perfectly. It can also observe the target position perfectly, if the target is visible. Uncertainties in control and sensing can be easily incorporated into the POMDP if needed (see Section III-B). If the robot reaches the current target position while the call status is on, it receives a reward of 100. The robot receives a reward of -1 for a horizontal or vertical move, a reward of $-\sqrt{2}$ for a diagonal move, and a reward of 0 if it stays stationary. The discount factor is set to 0.95.

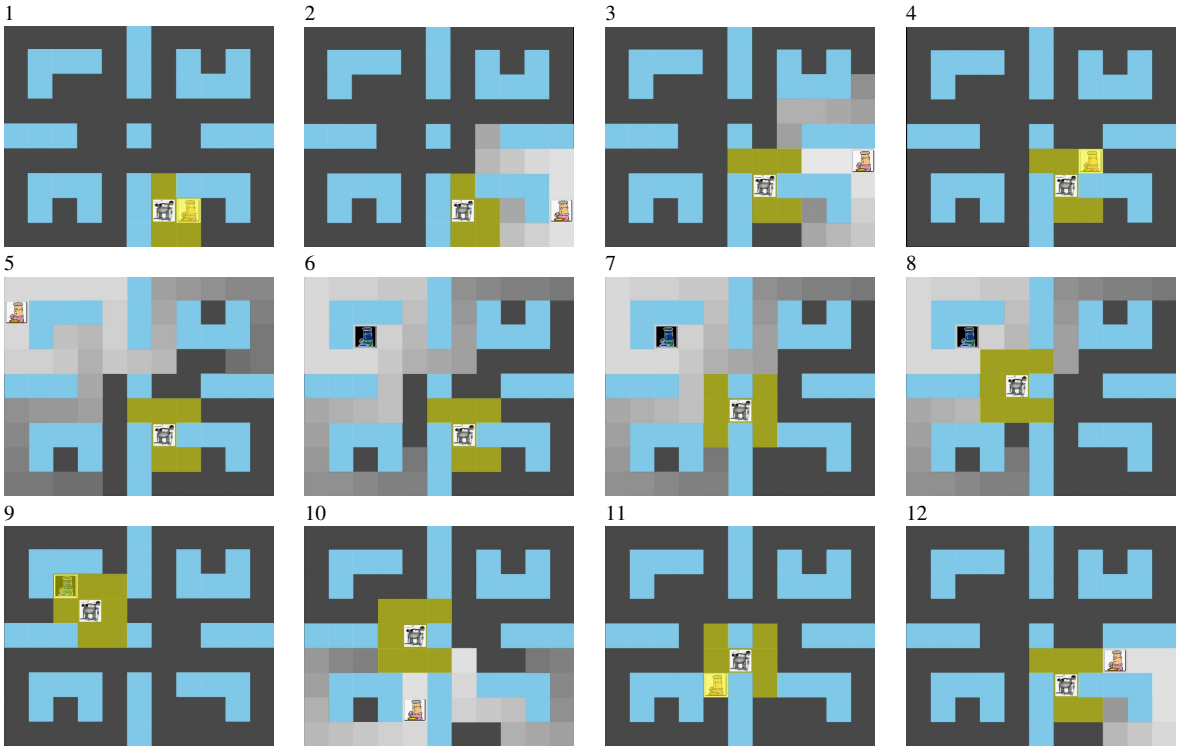


Fig. 4. Snapshots of a simulation run.

In the first experiment, we have a home-like environment (Fig. 3*a*). The corresponding POMDP has $9,248$ states. SARSOP computed a policy in about 48 minutes. We performed several simulation runs to examine its performance and observed interesting robot tracking behaviors:

- anticipatory moves that exploit target dynamics,
- information-gathering moves that reduce target position uncertainty,
- approaching the target along a nearly optimal path when the robot is called,
- minimizing movement by allowing the target to get out of sight, but not compromising long-term tracking performance.

It is important to bear in mind that these behaviors are not manually specified, but automatically captured by the POMDP through policy computation.

Snapshots of a single simulation run are shown in Fig. 4. Initially, the target lies within the robot sensor's visibility region, and the robot's belief on target position consists of a single peak (snapshot 1). As the target moves, the robot does not follow along and intentionally let the target get out of sight, in order to minimize movement and reduce energy consumption. Now, although the target is *not* visible, the robot still has the target reasonably well localized by maintaining a belief on the target position: the target is well within the high-probability region of the current belief (snapshot 2). Instead of following the target, the robot tries to anticipate the future position of the target by exploiting the target dynamics and makes a move towards this position (snapshot 3). As there is no call, the robot's move purely serves the purpose of gathering information on the target position. When the target passes by, the belief on target position is sharpened (snapshot 4). If the target is not observed for a while, the uncertainty may become large, but the robot is still able to maintain a belief that reflects the current target position well: the target is located within a high-probability region (snapshot 5). When there is a call (snapshot 6), it uses the current belief to find the region that contains the target with high probability. It then moves towards the region along the shortest path (snapshots 6–9). In general, the robot may need to search this region, but here it luckily finds the target right away and receives a high reward (snapshot 9). The robot then makes another anticipatory move to reduce target position uncertainty (snapshots 10–12). Interestingly, the robot position in snapshot 12 is exactly the same as that in snapshot 3, despite that the target positions and beliefs are quite different. It is, of course, not coincidence. This particular position guards both of the two ways into the lower right corner of the environment. By occupying this position, the robot can intercept the target as it exits the entrances without following it. The tracking behavior here reveals that the computed policy captures well the interaction between the environment geometry and the target dynamics. In this simulation run, there are 3 calls in total, and all are answered in time. The target travels a total distance of 141, and the robot travels a totals distance of about 20.

In the second experiment, the environment contains a special cell corresponding to a bathroom lying on the target's path (Fig. 3*b*). After entering the bathroom, the target stays there with probability 0.95 and leaves with probability 0.05 in each step. The corresponding POMDP has $7,200$ states. SARSOP computed a policy in about 16 minutes. Roughly,

to execute this tracking policy, the robot moves on the inner loop (the thick white curve in Fig. 3*b*) and follows the target that moves along the outer loop (the dashed black curve in Fig. 3*b*). It approaches the target directly when called.

Videos of both experiments above as well as additional experiments are available at `http://motion.comp.nus.edu.sg/projects/tracking/tracking.html`. We are currently performing more experiments to evaluate tracking performance quantitatively.

B. Computational Efficiency

We compared the performance of SARSOP and HSVI2, which, to our knowledge, has so far got the best experimental performance among the point-based POMDP algorithms. Both algorithms were implemented in C++ and tested on the same platform. For the two experiments above, HSVI2 took 63 minutes and 48 minutes, respectively, to compute policies of comparable quality.

We also compared SARSOP and HSVI2 on a set of standard tests for point-based POMDP algorithms, including ones on robot navigation and tracking, such as Hallway, Tiger Grid, Tag, *etc.*. The results show that SARSOP outperformed HSVI2 by many times on some problems, while remaining competitive on the rest. The improved performance can be attributed to the more aggressive α-vector pruning and, to some extent, belief point pruning. For lack of space, we refer to [9] for details on this experimental comparison.

The performance of SARSOP is affected by the δ parameter for α-vector pruning, but not sensitive to it. In all our target tracking experiments, the δ value for α-vector pruning was set to 1×10^{-2}. See [9] for experimental results on the dependency of SARSOP's performance on δ.

VI. Conclusion

In this work, we model target tracking as a POMDP, which unifies two tasks, target search and target following, that are often studied independently. To overcome the high computational complexity of solving POMDPs, we present a new point-based POMDP algorithm called SARSOP, which is based on successively approximating the space reachable under optimal policies. In simulation experiments, SARSOP successfully computed policies for tracking problems with more than 9,000 states. On a set of standard test problems, SARSOP outperformed the fastest existing point-based algorithm by many times on some problems, while remaining competitive on the rest. Along with other reports in literature [7], [8], [16], our results indicate that with the advances in POMDP solution algorithms, the POMDP approach is gradually becoming practical for non-trivial robot tasks.

We are currently conducting more experiments to evaluate the tracking performance of our POMDP algorithm quantitatively. We are also implementing SARSOP as a software package and expect to release it in the near future.

Acknowledgments. We thank Leslie Kaelbling and Tomas Lozano-Pérez from M.I.T. for many insightful discussions of POMDPs and target tracking. This work is supported in part by MoE AcRF grant R-252-000-327-112 and NUS ARF grant R-252-000-240-112.

References

[1] T. Bandyopadhyay, Y.P. Li, M.H. Ang Jr., and D. Hsu. A greedy strategy for tracking a locally predictable target among obstacles. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 2342–2347, 2006.

[2] M. Bennewitz, W. Burgard, G. Cielniak, and S. Thrun. Learning motion patterns of people for compliant robot motion. *Int. J. Robotics Research*, 24(1):31–48, 2005.

[3] B.P. Gerkey, S. Thrun, and G. Gordon. Visibility-based pursuit-evasion with limited field of view. *Int. J. Robotics Research*, 25(4):299–315, 2006.

[4] H.H. González-Baños, C.Y. Lee, and J.C. Latombe. Real-time combinatorial tracking of a target moving unpredictably among obstacles. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 1683–1690, 2002.

[5] L.J. Guibas, J.C. Latombe, S.M. LaValle, D. Lin, and R. Motwani. Visibility-based pursuit-evasion problem. *Int. J. Computational Geometry & Applications*, 9(5):471–494, 1999.

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

[7] 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.

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

[9] D. Hsu, W.S. Lee, and N. Rong. Accelerating point-based POMDP algorithms through successive approximations of the optimal reachable space. Technical Report TRA4/07, National University of Singapore. School of Computing, April 2007.

[10] 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. To appear.

[11] 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.

[12] S.M. LaValle, H.H. González-Baños, C. Becker, and J.C. Latombe. Motion strategies for maintaining visibility of a moving target. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 731–736, 1997.

[13] R. Murrieta-Cid, H.H. González-Baños, and B. Tovar. A reactive motion planner to maintain visibility of unpredictable targets. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 4242–4248, 2002.

[14] R. Murrieta-Cid, A. Sarmiento, and S. Hutchinson. A motion planning strategy to maintain visibility of a moving target at a fixed distance in a polygon. In *Proc. IEEE Int. Conf. on Advanced Robotics*, 2003.

[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] G. Shani, R. Brafman, and S. Shimony. Adaptation for Changing Stochastic Environments through Online POMDP Policy Learning. In *Proc. Eur. Conf. on Machine Learning*, pages 61–70, 2005.

[18] 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.

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

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

[21] I. Suzuki and M. Yamashita. Searching for a mobile intruder in a polygonal region. *SIAM Journal on Computing*, 21(5):863–888, 1992.

[22] R. Vidal, O. Shakernia, H.J. Kim, D.H. Shim, and S Sastry. Probabilistic pursuit-evasion games: theory, implementation, and experimental evaluation. *IEEE Trans. on Robotics & Automation*, 18(5):662–669, 2002.