A point-based POMDP algorithm for robot planning

Matthijs T. J. Spaan Nikos Vlassis
Informatics Institute, University of Amsterdam
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
{mtjspaan,vlassis}@science.uva.nl

***Abstract*— We present an approximate POMDP solution method for robot planning in partially observable environments. Our algorithm belongs to the family of point-based value iteration solution techniques for POMDPs, in which planning is performed only on a sampled set of reachable belief points. We describe a simple, randomized procedure that performs value update steps that strictly improve the value of all belief points in each step. We demonstrate our algorithm on a robotic delivery task in an office environment and on several benchmark problems, for which we compute solutions that are very competitive to those of state-of-the-art methods in terms of speed and solution quality.**

I. Introduction

As autonomous robots are being applied in more and more complex domains the need grows for tractable ways of planning under uncertainty. Classical motion planning [1] typically assumes that planning can be carried out without taking into account the uncertainty in the robot motion and the sensor observations. However, in order for a robot to execute its task well in a real-world scenario it has to deal properly with different types of uncertainty: a robot is unsure about the exact consequence of executing a certain action and its sensor observations are noisy. Robotic planning becomes even harder when different parts of the environment appear similar to the sensor system of the robot. Such "perceptual aliasing" is common in office environments where robots are often employed. In these partially observable domains a robot needs to explicitly reason with uncertainty in order to successfully carry out a given task, e.g., navigating through an office to deliver mail.

Partially observable Markov decision processes (POMDPs) provide a rich mathematical framework for solving such planning problems, with several applications in operations research [2], artificial intelligence [3], and robotics [4], [5], [6]. In a robotic context, a standard POMDP model assumes a discrete state, observation, and action space for the robot, and a reward that the robot gets in each time step, depending on the current task. The POMDP defines a sensor model specifying the probability of observing a particular sensor reading in a specific state, and a stochastic transition model which captures the uncertain outcome of executing an action.

In many situations a single sensor reading does not provide enough evidence to determine the complete and true state of the system. For instance, a robot in an office environment might detect that it is located in a corridor, but cannot tell in which one. If however the robot remembered from which room it entered the corridor, it would know which of the possible corridors it was observing. The POMDP framework allows for successfully handling such situations by defining and operating on the *belief state* of a robot. A belief is a probability distribution over all states and summarizes all information regarding the past. Solving a POMDP now means computing a policy—i.e., a mapping from belief states to actions—that maximizes the average collected reward of the robot in the task at hand. Such a policy prescribes for every belief state the action that maximizes the expected reward a robot can obtain in the future, and is optimal for a certain planning horizon.

Unfortunately, solving a POMDP in an exact fashion is an intractable problem [7], [8]. Intuitively speaking, looking one time step deeper into the future requires considering each possible action and each possible observation. As such, the cost of computing the solution for a POMDP grows exponentially with the desired planning horizon. To circumvent these intractability issues several approximate POMDP algorithms have been proposed in the literature [9], [10]. More specifically, in robotics a number of simple heuristic control strategies have been applied, most of which build on a solution of the underlying fully observable Markov decision process (MDP) [4], [5], [6].

An alternative line of research on approximate POMDP algorithms involves the use of a sampled set of *belief points* on which planning is performed [10], [11], [12], [13], [14]. The idea is that instead of planning over the complete belief space of the robot (which is intractable for large state spaces), planning is carried out only on a limited set of prototype beliefs that have been sampled by letting the robot interact with the environment. This gives rise to efficient algorithms that can handle state spaces in the order of hundreds of states, and that produce results better than the above MDP-based approximate POMDP algorithms.

In this paper we apply the randomized approximate value iteration algorithm of [14] to planning in a robotic context, which features large state spaces and high dimensional observations. Experimental results indicate that the algorithm can successfully handle large POMDPs, and is very competitive to other algorithms both in terms of solution quality as well as speed; in several benchmark problems it is at least one order of magnitude faster than state-of-the-art algorithms. We show results from an office delivery task involving a mobile robot with omnidirectional vision in a highly perceptually aliased office environment, where the number of possible robot locations is in the order of hundreds.

II. Planning in a POMDP framework

We first review more formally briefly the POMDP framework in the robotic planning context that we are discussing. A POMDP model describes the interaction of a robot with its environment as the iteration of the following two steps:

1) At any time step the environment is in a state $s \in S$. A robot that is embedded in the environment takes an action $a \in A$ and receives a reward $r(s,a)$ from the environment as a result of this action. The environment switches to a new state s' according to a known stochastic transition model $p(s'|s,a)$. The Markov property entails that s' only depends on s (and a). The transition model is typically assumed Gaussian when a is a movement action, reflecting errors in the robot motion.
2) The robot perceives an observation $o \in O$ that depends on its action. This observation provides the robot with information about the state s through a known stochastic observation model $p(o|s,a)$. In one of our experiments, for instance, the observations of the robot are features extracted from omnidirectional camera images.

All sets S, O, and A are assumed discrete and finite here (see [15] for the case when S and A are continuous). In order for a robot to choose its actions successfully in such a partially observable environment some form of memory is needed. In a belief state POMDP we maintain a discrete distribution $b(s)$ over states s that summarizes all information about the past. This distribution, or belief, is a Markovian state signal and can be updated using Bayes' rule each time the robot takes an action a and receives an observation o, as follows:

$$b_a^o(s') \propto p(o|s',a) \sum_s p(s'|s,a) b(s) \tag{1}$$

with $\sum_s b_a^o(s) = 1$. This belief update scheme is the workhorse of many robot localization techniques [16].

Mapping the original problem to a belief space—a simplex of dimension $|S|$—allows the use of dynamic programming techniques for fully observable Markov decision processes (MDPs). The planning task then becomes one of computing an optimal *policy*, a mapping from beliefs to actions that maximizes the expected discounted future reward of the robot $E[\sum_t^\infty \gamma^t r(s_t,a_t)]$, where γ is a discount rate, $0 \leq \gamma < 1$. The discount rate is used to ensure a finite sum and to prefer short trajectories to long ones: if there are two paths for a robot to reach a goal location we want it to prefer the fastest one. The discount rate is usually chosen close to 1.

A policy can be defined by a value function, which estimates the expected amount of future discounted reward for each belief state. The value function of an optimal policy is called the optimal value function and is denoted by V^*. It is a fixed point of the equation $V = HV$, with H the Bellman backup operator:

$$V(b) = \max_a \Big[\sum_s r(s,a) b(s) + \gamma \sum_{o,s,s'} p(o|s',a) p(s'|s,a) b(s) V(b_a^o) \Big] \tag{2}$$

where b_a^o is given by (1). We refer to [3], [10] for more technical details.

A. Value iteration in POMDPs

A classical method for solving POMDPs is value iteration. This method iteratively builds better estimates of V^* by applying the operator H to an initially piecewise linear and convex value function V_0 [2]. The intermediate estimates $V_1, V_2, \ldots$ will then also be piecewise linear and convex. We will throughout assume that a value function V_n at step n is represented by a finite set of vectors $\{\alpha_n^1, \alpha_n^2, \ldots\}$. Additionally, with each vector an action is associated, which is the optimal one to take in the current step, assuming optimal actions are executed in following steps. The value of a belief point b is then

$$V_n(b) = \max_{\alpha_n^i} b \cdot \alpha_n^i, \tag{3}$$

where $(\cdot)$ denotes inner product.

The main idea behind many value iteration algorithms for POMDP is that for a given value function V_n and a particular belief point b we can easily compute the vector α_{n+1}^b of HV_n such that

$$\alpha_{n+1}^b = \arg\max_{\alpha \in \{\alpha_{n+1}^i\}} b \cdot \alpha \tag{4}$$

where $\{\alpha_{n+1}^i, \forall i\}$ is the (unknown) set of vectors for HV_n. We will denote this operation $\alpha_{n+1}^b = \texttt{backup}(b, V_n)$. The difficult task, however, is to ensure that *all* vectors of HV_n are generated. Traditional algorithms [17], [3] search in the belief simplex for a minimal set of belief points $\{b_m\}$ that generate the necessary set of vectors for the new horizon value function:

$$\bigcup_{b_m} \texttt{backup}(b_m, V_n) = HV_n. \tag{5}$$

Unfortunately, such an approach can be very costly when the dimensionality of S is high, as it requires solving a number of linear programs.

B. Approximate techniques

One can also abandon the idea of doing exact value backups and settle for useful approximations. One simple heuristic technique commonly used is Q_{MDP} [9]. It treats the POMDP as if it were fully observable and solves the underlying MDP, e.g., using value iteration [18]. Then it uses the resulting $Q(s,a)$ values to define a control policy π as $\pi(b) = \arg\max_a \sum_s b(s) Q(s,a)$. The Q_{MDP} algorithm can be very effective in some domains, but the policies it computes will not take informative actions, as the Q_{MDP} solution assumes that any uncertainty regarding the state will disappear after taking one action.

In the recently introduced *point-based* techniques, a set of belief points are first sampled from the belief simplex by letting the robot interact with the environment, and then value updates are performed on these points only [10], [11], [13], [14]. In particular, the PBVI [13] algorithm samples a set B of belief points from the belief simplex (by stochastic

simulation), it repeatedly applies the backup operator on each $b \in B$ for a number of steps, then expands the set B, and so forth. Point-based solution techniques are justified by the fact that in most robotic problem settings the belief simplex is sparse, in the sense that only a limited number of belief points are ever reached when the robot directly interacts with the environment. In these cases, one would like to plan only for those reachable beliefs instead of planning over the complete belief simplex.

III. A RANDOMIZED POINT-BASED ALGORITHM

In [14] we proposed a simple approximate algorithm for solving POMDPs. We first let the robot randomly explore the environment and collect a set B of reachable belief points. We then initialize the value function V_0 as a single vector with all its components equal to $\frac{1}{1-\gamma}\min_{s,a} r(s,a)$. Starting with V_0, it performs a number of value function update steps until convergence. Given V_n, a value function update step is as follows:

Value function update

1) Set $V_{n+1} = \emptyset$. Set $\tilde{B} = B$.
2) Sample a belief point b from $\tilde{B}$ uniformly at random.
3) Compute $\alpha^b_{n+1} = \texttt{backup}(b, V_n)$. If $b \cdot \alpha^b_{n+1} \geq V_n(b)$ then add α^b_{n+1} to V_{n+1}, otherwise add $\alpha^b_n = \arg\max_{\alpha \in \{\alpha^i_n\}} b \cdot \alpha$ to V_{n+1}.
4) Compute $\tilde{B} = \{b \in B : V_{n+1}(b) < V_n(b)\}$. If $\tilde{B} = \emptyset$ then stop, otherwise go to 2.

The hope is that by randomly sampling belief points from (increasingly smaller) subsets of B we quickly build a function V_{n+1} that is an upper bound to V_n over B. In the first iterations where V_n and V_{n+1} differ substantially the random backup of points will allow for a quick build-up of an upper bound to V_n. As a result, the number of vectors generated in each backup step will be small compared to the size of B. Moreover, as the cost of backup depends linearly on the number of vectors of V_n, the first update steps can be very fast. This allows the method to quickly reach a good approximate solution with few vectors (see the experiments below).

The main difference with other point-based value update schemes is that we do not back up each $b \in B$, but instead back up (random) points until the value of every $b \in B$ has improved or remained the same. The intuition is that in this way we limit the growth of the number of vectors in the successive value function estimates. Moreover, we can afford to use a large set B of belief points sampled in advance, as opposed to other algorithms that propose growing B incrementally [11], [13].

IV. EXPERIMENTS

We will demonstrate how the presented algorithm can be applied to robotic planning problems which typically have to deal with many states, high dimensional sensor readings, perceptual aliasing, and uncertain actions. We will also briefly report on results comparing our algorithm on several benchmark problems from the POMDP literature, which are commonly used to test scalable POMDP solution techniques.

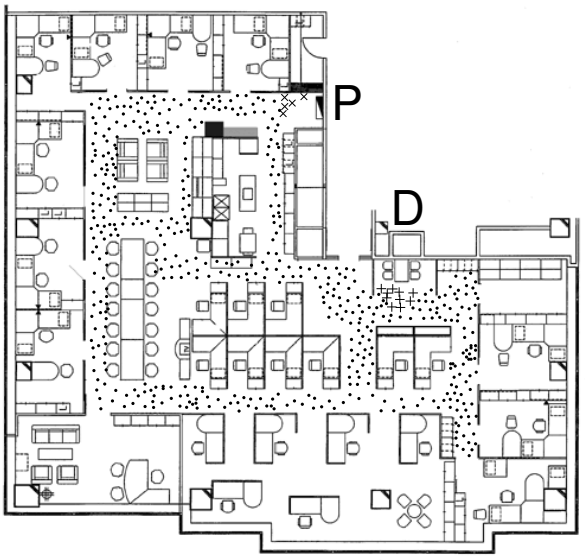


Fig. 1. The environment of the TRC domain. The markers $\{\cdot, +, \times\}$ denote the grid positions of the problem. The positions $\times$ (close to P) are the pickup locations, the ones marked by $+$ (below D) are the delivery (goal) positions.

A. The TRC domain

We applied our algorithm on a problem which is based on data obtained in a realistic setting. The TRC domain is a delivery task in an office environment with 1000 states. The task is to pick up mail at the entrance of the office (P, see Fig. 1) and deliver it to a certain room (D). At the start of the task the robot is uncertain about its location, and whether it has picked up the mail. The observation model is based on panoramic images taken by an omnidirectional camera mounted on the robot. We used the MEMORABLE[1] robot database that contains a set of approximately 8000 images collected manually by driving the robot around in a 17 $\times$ 17 meters office environment with constant orientation, with a sampling resolution of 0.1 meters. Fig. 2 shows some example images from this database.

As our algorithm assumes finite and discrete sets S and O we need to discretize the state space and the observation space. The state space is defined as the cross-product of the robot's location x and a single bit, indicating whether the robot has already successfully picked up the mail which needs delivery. For discretizing the positions in the map of the environment we performed a k-means clustering [19] on a subset of all possible positions, resulting in a grid of 500 positions $X = \{x_i\}$, which are depicted in Fig. 1.

For the discretization of the observation space one should choose the number of prototype observations with concern. A large number of observations can provide more discriminant information on the true position of the robot and thus lead to

[1]The MEMORABLE database has been provided by the Tsukuba Research Center in Japan, for the Real World Computing project.

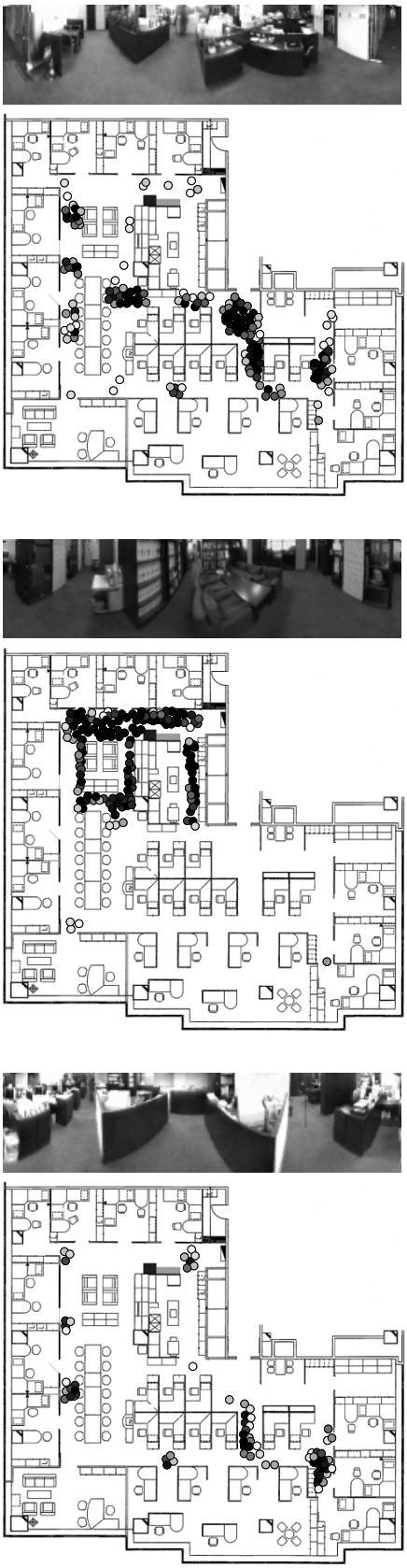


Fig. 2. Panoramic images corresponding to three prototype feature vectors $o_k \in O$, and below each one its induced $p(x|o_k)$. The darker the dot, the higher the probability.

more peaked beliefs. The more peaked the reachable beliefs are, the more the problem resembles the underlying MDP and the easier it would be to find a good policy. On the other hand, a large number of observations will not solve the problem of perceptual aliasing. Furthermore, the number of observation prototypes determines the size of the set $\{\alpha^i_{n+1}\}$ used in the `backup` operations in (4). As such, the number of observation prototypes increases the computational requirements of the algorithm. As in [20], we applied Principal Component Analysis (PCA) on the image data in order to reduce their dimensionality. We computed a three-dimensional feature vector for each one of a (randomly chosen) set of 1000 images, by projecting them to the first three eigenvectors (those with the largest eigenvalues) of their covariance matrix. Finally, to discretize this feature space, we used k-means clustering resulting in 10 three-dimensional prototype feature vectors $\{o_1, \ldots, o_{10}\}$.

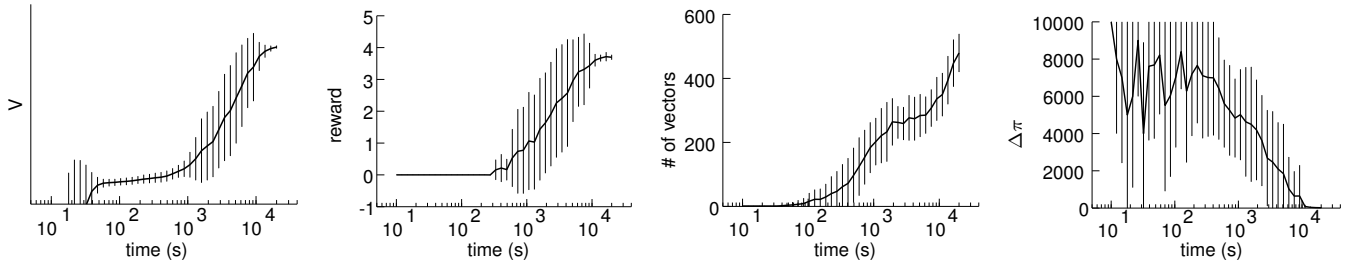
Then we constructed the discrete observation model $p(o|s)$ as follows (where we dropped the dependence on a for simplicity). We projected each image in the database to the feature space and found its nearest prototype feature vector among $\{o_k\}_{k=1\ldots10}$. We also associated each robot position in the database with its nearest prototype position in X. Then the probability of observing a prototype feature vector o_k from a prototype robot location x_k can be computed by a histogram operation as follows

$$p(o_k|x_k) = \frac{p(o_k, x_k)}{\sum_{o_l} p(o_l, x_k)} \tag{6}$$

where $p(o_k, x_k)$ is simply the fraction of pairs $\{o_i, x_i\}$ in the database such that $o_i \in o_k$ and $x_i \in x_k$, where '$\in$' denotes nearest-cluster membership. Finally, we duplicated this model for both instances of the bit indicating whether the mail has been picked up. Fig. 2 displays three panoramic images closest to three $o_k \in O$, and the corresponding $p(x|o_k)$.

The robot can execute four basic motion commands {*north, east, south, west*} which transports it according to a Gaussian distribution centered on the expected resulting position (translated two meters in the corresponding direction). In order to accomplish its task the robot must first execute the *pickup* action in one of the five pickup states near the entrance of the office (marked by P in Fig. 1). Only in one of these states the action results in flipping the pickup bit, after which delivering the mail has become possible. To complete the task and receive a reward of 10 the robot has to execute the *delivery* in one of the ten delivery states (indicated by D in Fig. 1). Trying to deliver without the mail or delivering to the wrong location is penalized with a reward of -10. Attempting to pick up the mail outside the pickup locations is penalized with reward -1. All motion actions yield no reward.

We sampled a belief set B of $10,000$ points by experiencing the environment: we let the simulated robot take random actions, apply Bayes' rule (1), record its belief at each step, and reset only when a successful delivery has been made. Only three deliveries were made during sampling and collecting all beliefs takes less than 20 seconds. The set B remains fixed



(a) Value. (b) Reward. (c) Nr. of vectors. (d) Policy changes.

Fig. 3. Results for the TRC domain.

during execution of the algorithm and during all experimental trials.

We ran the described algorithm 10 times with different random seeds. To evaluate the computed value function estimates we collected rewards by sampling trajectories from 100 random starting locations with the pickup bit off. Note that the robot has no knowledge regarding the pickup bit until it has picked up the mail at the appropriate location. In our experiments we used a discount factor $\gamma = 0.95$ and each trajectory was stopped after 100 steps (if the robot had not yet delivered the mail by then).

Fig. 3 shows the good performance of our algorithm, the error bars indicate standard deviation within the 10 runs of the algorithm. Fig. 3(a) displays the value as estimated on B, $\sum_{b \in B} V(b)$, (b) the expected discounted reward, (c) the number of vectors in the value function estimate, $|\{\alpha_n^i\}|$ and (d) the number of policy changes: the number of $b \in B$ that have a different optimal action in V_n compared to V_{n-1}. We can see that algorithm converges to approximately the same solution quality for all runs, both in value and collected reward. The amount of policy changes drops to below 0.5% of B, which can also be regarded as an indication of convergence. The number of vectors in the value function estimates grows as the planning horizon increases but the size of the value function remains acceptable. To visualize the policies that our algorithm computes, we plotted some example trajectories in Fig. 4. They show the computed policy directs the robot to first move to the pickup states, pick up the mail, and then move to the delivery locations in order to deliver the mail. We also tested Q_{MDP} on this problem, but it fails to compute a successful policy (it receives reward 0) due to the fact that it cannot represent the uncertainty regarding the pickup bit.

B. Benchmark problems

We will briefly report results on four problems from the POMDP literature, for more details we refer to [14]. Table I summarizes these problems in terms of the size of S, O and A. The Hallway, Hallway2 and Tiger-grid problems are maze domains previously used to test scalable POMDP solution techniques [9], [13]. The Tag domain is an order of magnitude larger than the first three problems, and models a search and tag game between two robots [13]. A summary of the results

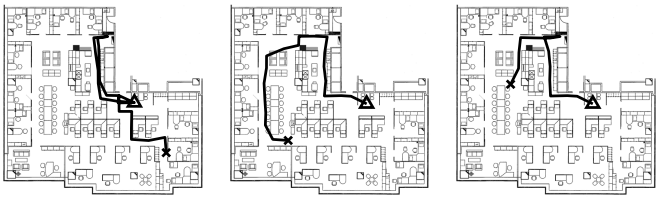


Fig. 4. Some example trajectories in the TRC environment. Start positions are marked with $\times$ and the last state of each trajectory is denoted by a $\triangle$.

TABLE I
POMDP DOMAIN CHARACTERISTICS.

Name	$\|S\|$	$\|O\|$	$\|A\|$
Tiger-grid	33	17	5
Hallway	57	21	5
Hallway2	89	17	5
Tag	870	30	5
TRC	1000	10	6

comparing our algorithm to PBVI, Q_{MDP} and BPI can be found in Table II. The bounded policy iteration (BPI) approach searches though the space of bounded-size, stochastic finite state controllers [21].

Our results are computed in Matlab on an Intel Pentium IV 2.4 GHz computer. PBVI results are taken from [13] and BPI results are quoted from [21], so time comparisons are rough. The results show that in the Tag problem our algorithm displays better control quality than Q_{MDP}, PBVI, and BPI, while it uses fewer vectors than the latter two. Our algorithm reaches very competitive control quality using a relatively small number of vectors.

V. CONCLUSIONS AND FUTURE WORK

We described in this paper a randomized approximate point-based value iteration algorithm for solving POMDPs, and demonstrated how to apply it to planning in a robotic context. The proposed algorithm performs value update steps, trying to upper bound in each iteration the current value function V_n, as estimated on a sampled set B of belief points. The main difference with other point-based value iteration algorithms is that we do not back up the value on each $b \in B$, but back

TABLE II

RESULTS ON BENCHMARK PROBLEMS.

Method	Reward	Vectors	Time (s)
Tiger-grid			
our	2.34	134	104
PBVI	2.25	470	3448
Q_{MDP}	0.23	n.a.	2.76
Hallway			
our	0.51	55	35
PBVI	0.53	86	288
Q_{MDP}	0.27	n.a.	1.34
Hallway2			
our	0.35	56	10
PBVI	0.34	95	360
Q_{MDP}	0.09	n.a.	2.23
Tag			
our	-6.17	280	1670
BPI	-9.18	940	59772
PBVI	-9.18	1334	180880
Q_{MDP}	-16.9	n.a.	16.1

up randomly selected points from B until the value of every $b \in B$ has improved (or at least remained the same). Running the algorithm on some benchmark problems from the literature indicated very competitive performance to similar algorithms: in most cases it is at least one order of magnitude faster than state-of-the-art algorithms, and equally good or better in terms of the quality of the computed policy.

We reported how the described algorithm can be applied to robotic planning problems. Robots typically have to deal with large state spaces, high dimensional sensor readings, perceptual aliasing and uncertain actions. We defined a mail delivery task in which a simulated robot has to deliver mail in an office environment. We proposed to perform PCA on the omnidirectional camera images the robot observes and perform clustering in the projected space to extract observation prototypes. We have shown our algorithm can successfully solve the resulting POMDP model.

As future work we would like to extend our algorithm to efficiently tackle POMDP problems with continuous state and/or action spaces. Furthermore, we would like to study multi-robot extensions of the algorithm.

Acknowledgments

This research is supported by PROGRESS, the embedded systems research program of the Dutch organization for Scientific Research NWO, the Dutch Ministry of Economic Affairs and the Technology Foundation STW, project AES 5414.

REFERENCES

[1] J. Latombe, *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[2] E. J. Sondik, "The optimal control of partially observable Markov decision processes," Ph.D. dissertation, Stanford University, 1971.

[3] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, pp. 99–134, 1998.

[4] R. Simmons and S. Koenig, "Probabilistic robot navigation in partially observable environments," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 1995, pp. 1080–1087.

[5] A. R. Cassandra, L. P. Kaelbling, and J. A. Kurien, "Acting under uncertainty: Discrete bayesian models for mobile robot navigation," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1996.

[6] G. Theocharous and S. Mahadevan, "Approximate planning with hierarchical partially observable Markov decision processes for robot navigation," in *Proc. IEEE Int. Conf. on Robotics and Automation*, Washington D.C., 2002.

[7] C. H. Papadimitriou and J. N. Tsitsiklis, "The complexity of Markov decision processes," *Mathematics of operations research*, vol. 12, no. 3, pp. 441–450, 1987.

[8] O. Madani, S. Hanks, and A. Condon, "On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems," in *Proc. 16th National Conf. on Artificial Intelligence*, Orlando, Florida, July 1999.

[9] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling, "Learning policies for partially observable environments: Scaling up," in *Proc. 12th Int. Conf. on Machine Learning*, San Francisco, CA, 1995.

[10] M. Hauskrecht, "Value function approximations for partially observable Markov decision processes," *Journal of Artificial Intelligence Research*, vol. 13, pp. 33–95, 2000.

[11] K.-M. Poon, "A fast heuristic algorithm for decision-theoretic planning," Master's thesis, The Hong-Kong University of Science and Technology, 2001.

[12] N. Roy and G. Gordon, "Exponential family PCA for belief compression in POMDPs," in *Advances in Neural Information Processing Systems 15*. Cambridge, MA: MIT Press, 2003.

[13] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for POMDPs," in *Proc. Int. Joint Conf. on Artificial Intelligence*, Acapulco, Mexico, Aug. 2003.

[14] N. Vlassis and M. T. J. Spaan, "A fast point-based algorithm for POMDPs," in *Benelearn 2004: Proceedings of the Annual Machine Learning Conference of Belgium and the Netherlands*, Brussels, Belgium, Jan. 2004, pp. 170–176, (Also presented at the NIPS 16 workshop 'Planning for the Real-World', Whistler, Canada, Dec 2003).

[15] S. Thrun, "Monte carlo POMDPs," in *Advances in Neural Information Processing Systems 12*, S. Solla, T. Leen, and K.-R. Müller, Eds. MIT Press, 2000, pp. 1064–1070.

[16] ——, "Probabilistic algorithms in robotics," *AI Magazine*, vol. 21, no. 4, pp. 93–109, Win 2000.

[17] H. T. Cheng, "Algorithms for partially observable Markov decision processes," Ph.D. dissertation, University of British Columbia, 1988.

[18] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[19] A. Likas, N. Vlassis, and J. J. Verbeek, "The global k-means clustering algorithm," *Pattern Recognition*, vol. 36, no. 2, pp. 451–461, Feb. 2003.

[20] N. Vlassis, B. Terwijn, and B. Kröse, "Auxiliary particle filter robot localization from high-dimensional sensor observations," in *Proc. IEEE Int. Conf. on Robotics and Automation*, Washington D.C., May 2002, pp. 7–12.

[21] P. Poupart and C. Boutilier, "Bounded finite state controllers," in *Advances in Neural Information Processing Systems 16*, S. Thrun, L. Saul, and B. Schölkopf, Eds., 2004.