An On-line Planner for POMDPs with Large Discrete Action Space: A Quantile-Based Approach

Erli Wang[1], **Hanna Kurniawati**[2], **Dirk P. Kroese**[1]
{e.wang2, hannakur, kroese}@uq.edu.au
[1] School of Mathematics and Physics [2] School of Information Technology and Electrical Engineering
The University of Queensland, Brisbane, QLD 4072, AUSTRALIA

Abstract

Making principled decisions in the presence of uncertainty is often facilitated by Partially Observable Markov Decision Processes (POMDPs). Despite tremendous advances in POMDP solvers, finding good policies with large action spaces remains difficult. To alleviate this difficulty, this paper presents an on-line approximate solver, called Quantile-Based Action Selector (QBASE). It uses quantile-statistics to adaptively evaluate a small subset of the action space without sacrificing the quality of the generated decision strategies by much. Experiments on four different robotics tasks with up to 10,000 actions indicate that QBASE can generate substantially better strategies than a state-of-the-art method.

Introduction

Finding the optimal solution of a POMDP problem (Sondik 1971) is computationally intractable (Papadimitriou and Tsitsiklis 1987). Loosely speaking, four components lead to the high computational complexity in solving POMDP problems: long planning horizons and large state, action, and observation spaces. The past decade has seen significant advances in approximate POMDP solvers, enabling us to compute good decision strategies for POMDPs with large state spaces (Kurniawati and Yadav 2013; Kurniawati, Hsu, and Lee 2008; Luo et al. 2016; Pineau, Gordon, and Thrun 2003; Porta et al. 2006; Shani, Brafman, and Shimony 2007; Silver and Veness 2010; Smith and Simmons 2004; 2005; Somani et al. 2013) and large observation spaces (Bai, Hsu, and Lee 2014; Hoey and Poupart 2005), as well as up to a hundred look-ahead planning steps (Kurniawati et al. 2011).

Despite these advances, determining good decision strategies for problems with large action spaces remains difficult. A POMDP solver must compute, for each belief, an action that maximizes the expected total return with respect to the belief, even though evaluating this expectation itself is computationally expensive. Most successful solvers resort to enumerating all possible actions. When the action space is large, such enumeration is no longer feasible.

QBASE extends the Cross-Entropy method (Rubinstein and Kroese 2004) to partially enumerate the action space, so as to avoid full enumeration of the action space at each belief without sacrificing the quality of the resulting strategy too much. A small subset of actions is constructed by actions in top ϱ-quantile of the current estimate of the expected total return and actions drawn by uniform sampling without replacement. This allows more computational resources to be allocated to obtain more refined estimates for actions that are likely to perform well, as well as ensuring that all actions will eventually be visited. It has shown significant improvements compared to state-of-the-art method, POMCP (Silver and Veness 2010), for problems with large action space.

Background and Related Work

A POMDP is described by a tuple $\langle S, A, O, T, Z, R, b_0, \gamma \rangle$, where S is the set of *states*, A is the set of *actions*, and O is the set of *observations*. At each step, the agent is in some hidden state $s \in S$, takes an action $a \in A$, and moves from s to another state $s' \in S$ according to a transition density $T(s, a, s') = f(s' \mid s, a)$. The new state s' is then partially revealed via an observation o drawn from a conditional probability density $Z(s', a, o) = f(o \mid s', a)$ that represents uncertainty in sensing. After each step, the agent receives a reward $R(s, a)$, if it takes action a from state s. Due to the uncertainty in the effect of action and in sensing, the agent never knows its exact state. Instead, it maintains an estimate of its current state in the form of a *belief* b, which is a probability density on S.

The objective of a POMDP agent is to maximize its expected total reward —called *value function*—, by following at each time step the best *policy*— a mapping from beliefs to actions. Each policy induces a value function V, and the best policy gives rise to a value function V that satisfies:

$$V(b) = \max_{a \in \mathcal{A}} \underbrace{\left[\sum_{s \in \mathcal{S}} R(s,a) b(s) + \gamma \sum_{o \in O} f(o \mid b, a) V(\tau(b,a,o)) \right]}_{\text{Q-value function: } Q(b,a)}$$

where $f(o \mid b, a)$ represents the probability density of perceiving observation $o \in O$ after performing action $a \in A$ from belief b, and is computed as $\sum_{s', s \in S} f(o \mid s', a) f(s' \mid s, a) b(s)$. The notation $\tau(b, a, o)$ represents the new belief of the agent, after it performs action a and perceives observation o, and is computed as $\tau(b, a, o)(s') \propto \sum_{s \in S} Z(s', a, o)\, T(s, a, s')\, b(s)$. When the planning horizon is infinite, to ensure the problem is well

defined, rewards at subsequent time steps are discounted by a factor $0 < \gamma < 1$.

Finding the best action to perform from a belief is a stochastic optimization problem. Furthermore, since computing a good estimate of the Q-value function is costly, optimization methods that rely on gradients would be expensive to compute. Methods to alleviate this difficulty for continuous action MDP —the fully observable version of POMDP— have been proposed (Mansley, Weinstein, and Littman 2011). While for POMDP, GPS-ABT (Seiler, Kurniawati, and Singh 2015) alleviates the problem via Generalized Pattern Search. However, its convergence relies on a continuity property of the gradient of the value function, which is unlikely to be satisfied for general POMDP problems.

In this paper, we will relax the continuity requirement via a quantile-based approach. In particular, we are motivated by recent advances in solving Multi-Armed Bandit (MAB) problems (Chaudhuri and Kalyanakrishnan 2017; Wang, Kurniawati, and Kroese 2017) that indicates a quantile-based sampling approach can significantly outperform established methods when the number of arms is large.

Quantile-Based Action Selector (QBASE)

A. Overview and Belief Tree A brief summary of QBASE is presented in Algorithm 1. Given a POMDP model or a corresponding black-box simulator $\mathcal{P}$, we run QBASE with parameters $\langle \varrho, N_s, K, \beta \rangle$ (described in the next subsection). At each step it aims to compute the best action to perform from the current belief b for a fixed time limit, executes this action, and updates its belief to $b' = \tau(b, a, o)$, where $a \in A$ is the computed best action and $o \in O$ is the observation the agent perceives right after performing a from b. The process then repeats from the new belief b', until a termination condition is met.

To find the best action to perform from a belief, QBASE constructs a *belief tree*, denoted as $\mathcal{T}$. A belief tree is a tree where the nodes are sampled beliefs. For compactness, we denote the nodes of $\mathcal{T}$ and the beliefs they represent in the same way. An edge labeled (a, o) from a belief b to a belief b' in $\mathcal{T}$ means there is an action $a \in A$ and an observation $o \in O$ such that $b' = \tau(b, a, o)$. To construct the belief tree $\mathcal{T}$, QBASE follows a strategy similar to POMCP (Silver and Veness 2010).

QBASE represents each sampled belief as a set of particles (states) and estimates the value of each sampled belief via Monte Carlo backup. The best action is then the action that induces the best estimated value. QBASE estimates the Q-value $Q(b, a)$ as

$$b.\widehat{Q}(a) = \frac{1}{\left|H_{(b,a)}\right|} \sum_{h \in H_{(b,a)}} V(h, l). \tag{1}$$

Here, $H_{(b,a)} \subseteq H$ is the set of histories that correspond to paths in $\mathcal{T}$ that starts from the root node, pass through node b and then follow action a. Also, $l = l(b)$ denotes the depth level of node b in $\mathcal{T}$. The function $V(h, l)$ is the value of a history h starting from the l^{th} element, and is computed as $\left(\sum_{i=l}^{|h|} \gamma^{i-l} R(h_i.s, h_i.a)\right) + E_h$, where γ is the discount factor and R is the reward function. E_h is an estimate of the value of the last state in h, and is computed as the total discounted reward of a random walk for a pre-defined number of steps from this last state.

Algorithm 1: QBASE

Function QBASE*(b_0, $\mathcal{P}$, $\langle \varrho, N_s, K, \beta \rangle$)*:
  Initialize $\mathcal{T}$ with b_0 as the root node; INITINODE(b_0)
  while *running* **do**
    while *there is still time for planning* **do**
      GROWTREE ($\mathcal{T}$, $\mathcal{P}$, $\langle \varrho, N_s, K, \beta \rangle$)
    Perform action a, such $a = \arg\max_{a \in A} b.\mathbf{P}(a)$
    $o =$ get observation
    $b = \tau(b, a, o)$ `// update belief`

Function INITINODE*(b)*:
  for $a \in \mathcal{P}.A$ **do** $b.N(a) = 0$; $b.\mathbf{P}(a) = \frac{1}{|\mathcal{P}.A|}$
  $b.A_s =$sample N_s actions uniformly at random from A
  for $a \in b.A_s$ **do** $b.\mathbf{P}_s(a) = \frac{b.\mathbf{P}(a)}{\sum_{a \in b.A_s} b.\mathbf{P}(a)}$

Function GROWTREE*($\mathcal{T}$, $\mathcal{P}$, $\langle \varrho, N_s, K, \beta \rangle$)*:
  Set $d = 0$, $h = \emptyset$, $b =$ root of $\mathcal{T}$, $I_{rollout} =$ **false**
  $s \sim b$
  while $\gamma^d > \varepsilon$ *and* $I_{rollout} ==$ **false do**
    $a =$ SAMPLEACT(b, $\langle \varrho, N_s, K, \beta \rangle$)
    $(s', o, r) =$ SIMULATOR($\mathcal{P}, s, a$)
    Append $\langle s, a, o, r \rangle$ to h
    Associate $\langle s, a, o, r \rangle$ with b
    $b.particles = b.particles \cup \{s\}$
    $b.A_v = b.A_v \cup \{a\}$
    $s = s'$; d++ ; $b.N(a)$++ ; $b.N$++ ; $b' = \tau(b, a, o)$
    if b' *is not in* $\mathcal{T}$ **then**
      add b' as a child of b and set $I_{rollout} =$ **true**
    $b = b'$
  if $\gamma^d > \varepsilon$ **then**
    $E_h = 0$
  else
    INITINODE(b) ; $b.particles = \{s\}$
    $E_h =$ROLLOUT-POLICY(s)
  Append $\langle s, -, -, E_h \rangle$ to h
  Associate $\langle s, -, -, E_h \rangle$ with b
  UPDATEVALUES($\mathcal{T}$,h,E_h)

Function SAMPLEACT *(b, $\langle \varrho, N_s, K, \beta \rangle$)*:
  if $b.N > 0$ *and* $(b.N \bmod K) == 0$ **then**
    $\mathcal{E} =$ the top $\lfloor \varrho \cdot |A| \rfloor$ elements of sorted $b.\widehat{Q}$
    $b.A_s = \mathcal{E}$
    while $|b.A_s| < N_s$ **do**
      $a \sim \mathcal{U}(A \backslash \mathcal{E})$; Set $b.A_s = b.A_s \bigcup \{a\}$
    $m = \min_{a \in b.A_v} b.\widehat{Q}(a)$
    $M = \max_{a \in b.A_v} b.\widehat{Q}(a)$
    for $a \in b.A_v$ **do** $\mathbf{W}(a) = \boldsymbol{\alpha}_b(a) \frac{b.\widehat{Q}(a) - m}{M - m}$
    for $a \in b.A_v$ **do** $b.\mathbf{P}(a) = \frac{|b.A_v|}{|A|} \frac{\mathbf{W}(a)}{\sum_{a \in b.A_v} \mathbf{W}(a)}$
    for $a \in b.A_s$ **do** $b.\mathbf{P}_s(a) = \frac{b.\mathbf{P}(a)}{\sum_{a \in b.A_s} b.\mathbf{P}(a)}$
  $a \sim b.\mathbf{P}_s$

B. Sampling Actions Key to QBASE is the way it samples actions when generating histories to construct the belief tree $\mathcal{T}$. Given a belief node to expand, QBASE maintains a probability distribution function over the action space A and uses this distribution to sample actions, so as to avoid full enumeration of the action space at the beginning.

The question is, given a belief b, what distribution should QBASE use to sample the action space A? Ideally, we want to assign high probability mass to good actions and zero mass to bad actions. Of course, which actions are good and which are bad are a priori unknown, as otherwise the problem would have been solved. Therefore, QBASE aims to adaptively construct a distribution proportional to the Q-value function, interleaving improvement of the estimation of Q-values of sampled actions with adaptation of the distribution in a Cross-Entropy method fashion.

QBASE adapts the distribution in two stages. First, it identifies a subset of the action space, denoted as $b.A_s$ (line 37–40 of SAMPLEACT in Algorithm 1), where it expects to find good actions. At any given time, QBASE focuses on evaluating actions in this subset only. This set has a predefined size (denoted as N_s) and consists of two components: The top ϱ-quantile of A and $N_s - \varrho|A|$ exploration components, sampled uniformly without repetition from A without the exploitation components. The latter component ensures that all actions will eventually be evaluated.

In the second stage, QBASE assigns probability mass function (pmf) to actions in the subset $b.A_s$ (line 41–45 of SAMPLEACT). To ensure that the subset $b.A_s$ is evaluated sufficiently, the subset and its associated pmf will only be updated in batch mode, after $b.A_s$ has been evaluated for a fixed budget (parameter K of QBASE). Furthermore, QBASE aims to keep the trend in the pmf of actions in $b.A_s$ to be similar to probability over A, as this probability reflects QBASE's approximation on the relative differences in Q-values of the different actions.

To this end, after each batch, QBASE updates the probability over A based on recent estimates of Q-value, via a slight modification of the *proportional Cross-Entropy* (pCE) update (Goschin, Weinstein, and Littman 2013):

$$b.\mathbf{P}(a) \;\propto\; \boldsymbol{\alpha}_b(a)\,\frac{b.\widehat{Q}(a) - m}{M - m}. \tag{2}$$

The smoothing parameter $\boldsymbol{\alpha}_b(a)$ quantifies how much QBASE will trust the new estimate. This increases with the number of visits according to $\boldsymbol{\alpha}_b(a) = b.N(a)/(b.N(a) + \beta)$, where $b.N(a)$ is the number of visits for the pair (b, a) and β is a constant parameter. The quantities m and M refer to the minimum and maximum of the estimated Q-values up to this batch.

The probability over $b.A_s$, $b.\mathbf{P}_s(a)$ for $a \in b.A_s$ is then set to be proportional to the corresponding $b.\mathbf{P}(a)$ (line 45 of SAMPLEACT). Furthermore, given belief b, QBASE selects actions to evaluate by sampling $b.A_s$ using the probability function $b.\mathbf{P}_s(a)$.

C. Properties of QBASE An implicit assumption of QBASE is that good solutions are abundant. Abundance of solutions is a main reason why sampling-based algorithms work well in practice (Motwani and Raghavan 2010). Such an assumption is also common in motion planning (Choset 2005). This requirement is more general than the continuity property (i.e., nearby actions will lead to similar results).

Now, the question is whether QBASE can yield the optimal policy. QBASE asymptotically converges to near optimal solution. The probability $b.\mathbf{P}(a)$ converges to the scaled estimated Q-value of each action (Goschin, Weinstein, and Littman 2013), since $\boldsymbol{\alpha}_b(a)$ asymptotically converges to 1 for any node b in $\mathcal{T}$ and each action a from b. This means, executing the action with the highest probability is asymptotically equivalent to executing the action with the highest estimated Q-value. The error of estimated Q-value at root is bounded for finite horizon problems; this result can then be extended to the infinite-horizon case, similar to (Kearns, Mansour, and Ng 2002).

Results and Discussion

We compare QBASE with the state-of-the-art on-line POMDP solver, POMCP (Silver and Veness 2010). For a fair comparison, we ran POMCP using the authors' code and implemented QBASE in the POMCP code framework using C++. All experiments were run as a single thread process on an Intel Xeon E5-2620 v4 @ 2.10GHz with 128GB RAM.

To set the parameters for both methods and scenarios, we perform 20 preliminary runs for each scenario and use the parameters that generate the best results [1]. To estimate the quality of the policy generated, for each scenario, we ran each method $1,000$ times with best performing parameters.

A. Scenarios We briefly describe the test scenarios. Details are available in the Appendix of the expanded version of the paper (downloadable from `http://robotics.itee.uq.edu.au/~hannakur/dokuwiki/papers/icaps18_qbase.pdf`).

RockSample(n, k) (Smith and Simmons 2004) is a well-known benchmark for POMDP solvers. A robot must explore an environment of size $n \times n$, populated by k rocks. The goal of the robot is to sample as many good rocks as possible as fast as possible.

Navigation(d, n): An agent must navigate to a goal location in a d-dimensional grid world populated by obstacles, where each dimension is discretized into n cells. The agent initial position is not exactly known, but it must be at one of the 3^d cells in the corner. The agent can move to adjacent cells, within 3 cells away from its current position, resulting in 7^d possible actions. Its motion is accurate 90% of the time. The rest of the time, it reaches among the $7^d - 1$ remaining cells with equal probability. The agent can only observe the existence and position of walls surrounding its current cell, forming 2^{2d} observations. The observation function is perfect. However, since multiple states can generate the exact same observation, this scenario is partially observable. The agent receives a 1,000 reward for reaching the terminal states and incurs a -1 penalty for every movement.

Hunting(n, u, v)**:** Centralized control of multiple (u) robots, to catch multiple (v) targets moving in a grid-world of size

[1] Details of the parameters are in the expanded version

Table 1: Simulation Results

Scenario	t (s)	Method	Reward
Nav(2, 30), $\|A\| = 49$	1	POMCP	732 ± 5.0
$\|S\| \approx 10^2, \|Z\| = 16$		QBASE	742 ± 4.7
Nav(3, 30), $\|A\| = 343$	2	POMCP	561 ± 7.6
$\|S\| \approx 10^4, \|Z\| = 64$		QBASE	633 ± 8.6
Nav(4, 30), $\|A\| = 2,401$	5	POMCP	-8 ± 4.9
$\|S\| \approx 10^5, \|Z\| = 256$		QBASE	91 ± 10.9
RS(7, 8), $\|A\| = 13$	1	POMCP	18 ± 0.4
$\|S\| = 12,544, \|Z\| = 3$		QBASE	19 ± 0.4
RS(20, 50), $\|A\| = 55$	2	POMCP	18 ± 0.6
$\|S\| \approx 10^{17}, \|Z\| = 3$		QBASE	20 ± 0.7
RS(20, 100), $\|A\| = 105$	5	POMCP	14 ± 0.8
$\|S\| \approx 10^{32}, \|Z\| = 3$		QBASE	15 ± 0.9
HS(11, 2, 2), $\|A\| = 100$	1	POMCP	-72 ± 5.7
$\|S\| \approx 10^8, \|Z\| = 4$		QBASE	-70 ± 4.3
HS(11, 3, 3), $\|A\| = 1,000$	5	POMCP	-179 ± 7.0
$\|S\| \approx 10^{12}, \|Z\| = 8$		QBASE	-94 ± 5.7
HN(11, 2, 2), $\|A\| = 100$	1	POMCP	42 ± 2.9
$\|S\| \approx 10^8, \|Z\| = 4$		QBASE	45 ± 3.7
HN(11, 3, 3), $\|A\| = 1,000$	5	POMCP	26 ± 8.6
$\|S\| \approx 10^{12}, \|Z\| = 8$		QBASE	96 ± 7.3
HN(11, 4, 4), $\|A\| = 10,000$	10	POMCP	$-1,573 \pm 33.7$
$\|S\| \approx 10^{16}, \|Z\| = 16$		QBASE	68 ± 7.1

* *Nav* refers to *Navigation*; *RS* refers to *RockSample*; *HN* refers to *Hunting-Normal*; *HS* refers to *Hunting-Smart*

$n \times n$, populated by obstacles. At the beginning, the targets position are unknown, and represented as uniform distributions over the free cells. At each step, each robot can stay where it is, move to one of the 8 cells adjacent to its current position, or catch a target. Their motion has no error. Furthermore, at each step, each robot can perfectly detect whether there is target(s) located in the same cell as itself or in one cell to its North, South, East, or West directions. Note that although its detection is perfect, a robot cannot distinguish which target is being detected nor the exact position (out of the five cells) of the target. A small penalty -1 is imposed on movement action for each robot. The 'catch' action yields a $+100$ reward if the agent is in the same cell as the target(s), otherwise the action incurs a penalty of -100. The targets know exactly the positions of the robots, with two behaviour variation to avoid being captured:

- *Hunting-smart*: When a robot and a target occupy the same cell, the target can still get away, unless the robot performs the action 'catch'. This is the same as the target's behavior in *Tag* (Pineau, Gordon, and Thrun 2003).
- *Hunting-normal*: Once a robot and a target are in the same cell, the target cannot escape.

B. Results Table 1 presents the average expected discounted total reward with 95% confidence interval of the $1,000$ simulation runs for each scenario and method. Overall, QBASE outperforms POMCP in all test scenarios. Furthermore, in general, except for ***RockSample***, the gap between QBASE and POMCP increases, as the size of the problem increases. In ***RockSample***, the action spaces are relatively small, that the extra computation of constructing a subset via the quantile-based method that QBASE performs becomes an unnecessary overhead.

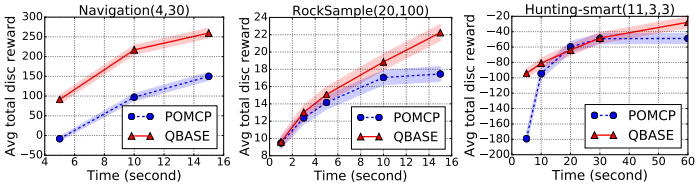


Figure 1: Performance with different planning time per step.

Figure 1 shows how the planning time per step affects performance in some of the scenarios. These trends are common in other scenarios too. The results indicate that, in general, for a small planning time, both POMCP and QBASE perform equally but, as we allow additional planning time, QBASE can significantly improve on POMCP. The reason is that, when the planning time is limited, both POMCP and QBASE do not have enough time to compute good Q-value estimates and can only build relatively shallow belief trees, causing both to perform equally poorly. However, when more time is allowed and deeper trees can be built, POMCP still needs to sweep the entire action space every time a node is added to the belief tree, which reduces the time it can spend on evaluating good actions. In contrast, QBASE evaluates only a small subset of the action space, guided by quantile-statistics, and can identify faster which actions are more promising and, as a result, can spend more resources on evaluating these actions.

A slightly different behavior is shown in ***Hunting-smart***$(11, 3, 3)$. In this scenario, with 5 seconds planning time, QBASE outperforms POMCP, but POMCP catches up at 20 seconds planning time, before being outperformed by QBASE again as more planning time per step is allowed. The reason is that this problem has a considerably large action space, and therefore POMCP's sweeping of the entire action space already takes a significant portion of the 5 seconds planning time, causing POMCP to perform badly. However, as more time is allowed, POMCP starts to improve its base-line performance into a relatively good and easy to find policy, and then plateaus at this policy. In contrast, QBASE can quickly identify good actions and generate this relatively good and easy to find policy fast, and then takes significant additional time to improve this policy further.

Conclusion

This paper introduces QBASE, a novel on-line approximate POMDP solver for problems with large discrete action spaces. It applies quantile statistics to adaptively construct a much smaller subset of the action space, so that more resources can be given to evaluate the Q-values of more promising actions. Experimental results on a range of robotics scenarios with action spaces up to 10,000 indicate that QBASE outperforms a state-of-the-art method.

We hope this new advancement in solving problems with large action spaces will further advance the practicality of POMDPs and allow more widespread applications of this robust approach to decision making.

Acknowledgments

We thank anonymous reviewers for their helpful comments. This work was supported by the Australian Research Council Centre of Excellence for Mathematical and Statistical Frontiers (ACEMS) under grant number CE140100049. Computational analyses were carried out in the Robotics Design Lab at the University of Queensland (UQ). Erli Wang would also like to acknowledge the support from UQ through the UQ International Scholarships scheme.

References

Bai, H.; Hsu, D.; and Lee, W. S. 2014. Integrated perception and planning in the continuous space: A POMDP approach. *IJRR* 33(9):1288–1302.

Chaudhuri, A. R., and Kalyanakrishnan, S. 2017. PAC identification of a bandit arm relative to a reward quantile. In *AAAI*, 1777–1783.

Choset, H. M. 2005. *Principles of robot motion: theory, algorithms, and implementation*. MIT press.

Goschin, S.; Weinstein, A.; and Littman, M. L. 2013. The Cross-Entropy method optimizes for quantiles. In *ICML (3)*, 1193–1201.

Hoey, J., and Poupart, P. 2005. Solving POMDPs with continuous or large discrete observation spaces. In *IJCAI*, 1332–1338.

Kearns, M.; Mansour, Y.; and Ng, A. Y. 2002. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine learning* 49(2):193–208.

Kurniawati, H., and Yadav, V. 2013. An Online POMDP Solver for Uncertainty Planning in Dynamic Environment. In *ISRR*.

Kurniawati, H.; Du, Y.; Hsu, D.; and Lee, W. 2011. Motion planning under uncertainty for robotic tasks with long time horizons. *IJRR* 30(3):308–323.

Kurniawati, H.; Hsu, D.; and Lee, W. S. 2008. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *RSS*.

Luo, Y.; Bai, H.; Hsu, D.; and Lee, W. S. 2016. Importance sampling for online planning under uncertainty. In *WAFR*.

Mansley, C. R.; Weinstein, A.; and Littman, M. L. 2011. Sample-based planning for continuous action Markov decision processes. In *ICAPS*.

Motwani, R., and Raghavan, P. 2010. *Randomized algorithms*. Chapman & Hall/CRC.

Papadimitriou, C. H., and Tsitsiklis, J. N. 1987. The complexity of Markov Decision Processes. *Math. of Operation Research* 12(3):441–450.

Pineau, J.; Gordon, G.; and Thrun, S. 2003. Point-based Value Iteration: An anytime algorithm for POMDPs. In *IJCAI*.

Porta, J. M.; Vlassis, N.; Spaan, M. T.; and Poupart, P. 2006. Point-based value iteration for continuous POMDPs. *Journal of Machine Learning Research* 7:2329–2367.

Rubinstein, R. Y., and Kroese, D. P. 2004. *The Cross-Entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer.

Seiler, K.; Kurniawati, H.; and Singh, S. 2015. An Online and Approximate Solver for POMDPs with Continuous Action Space. In *ICRA*.

Shani, G.; Brafman, R. I.; and Shimony, S. E. 2007. Forward search value iteration for POMDPs. In *IJCAI*, 2619–2624.

Silver, D., and Veness, J. 2010. Monte-Carlo planning in large POMDPs. In *NIPS*, 2164–2172.

Smith, T., and Simmons, R. 2004. Heuristic search value iteration for POMDPs. In *UAI*.

Smith, T., and Simmons, R. 2005. Point-based POMDP algorithms: Improved analysis and implementation. In *UAI*.

Somani, A.; Ye, N.; Hsu, D.; and Lee, W. S. 2013. DESPOT: Online POMDP planning with regularization. In *NIPS*, 1772–1780.

Sondik, E. J. 1971. *The optimal control of partially observable Markov processes*. Ph.D. Dissertation, Stanford University.

Wang, E.; Kurniawati, H.; and Kroese, D. P. 2017. CEMAB: A Cross-Entropy-based method for large-scale multi-armed bandits. In *Australasian Conference on Artificial Life and Computational Intelligence*, 353–365. Springer.

Appendix A. Parameter Settings

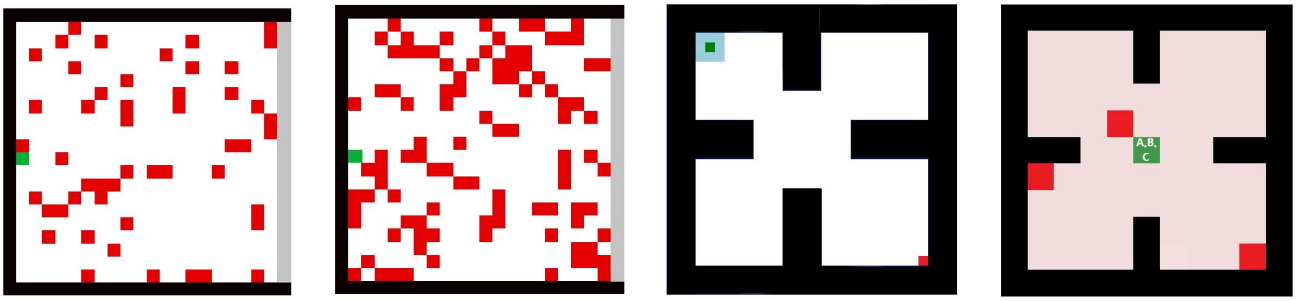
To set the parameters of QBASE and POMCP, we first carry out a set of pilot runs for each solver to determine the best one. Specifically, in QBASE, the largest subset N_s is determined by $\min(0.5\,|A|, 100)$, ϱ is selected from $\{0.3, 0.5, 0.7\}$, K from $\{1, 2\}$, and β from $\{5, 10, 20\}$. For POMCP, the exploration constant C is selected from $\{0.1, 1, 10, 100, 1,000, 10,000\}$.

Other parameters of the algorithm are set independently from problem domains. As the scale of the test problems is large, we set the discount factor $\gamma = 0.98$ and the tolerance of the approximate Q-value to $\varepsilon = 0.01$, in order to obtain a relative long planning horizon. As a result, the effective horizon is about $D = \log(\varepsilon)/\log(\gamma) \approx 228$. For a fair comparison, both solvers use the same rollout policy.

Appendix B. Details of Scenarios

B1. RockSample(n, k)

Figure 2(a) and (b) illustrate the scenarios for $(20, 50)$ and $(20, 100)$. Rocksample (Smith and Simmons 2004) is a well-known benchmark for POMDP solvers. A robot must explore an environment of size $n \times n$, populated with k rocks (marked as red squares). The position of the rocks are known exactly, but whether a rock is good or bad is unknown. In fact, at the beginning, each rock has a 0.5 chance of being good or bad. The goal of the robot is to sample as many good rocks as possible as fast as possible. The state space is the Cartesian product of the robot's position and the quality of the rocks, forming a state space of size $n^2 \cdot 2^k$. The robot can move to its North, South, East, and West cell,



(a) (b) (c) (d)

Figure 2: Illustration of some of the test scenarios: (a) ***RockSample***$(20, 50)$. (b) ***RockSample***$(20, 100)$. (c) ***Navigation***$(2, 30)$. (d) ***Hunting***$(11, 3, 3)$.

sample a rock at its current location, or remotely check rock $i = 1, \ldots, k$ to gain more information on whether it is good or bad. Its motion is perfect and the robot's position is fully observed. Checking a rock means that the robot applies a scanner to identify if the rock is good or bad. The reliability of the signal received decreases exponentially with the distance between the robot and the rock. The robot receives a reward of $+10$ if it samples a good rock or if it exits the environment (entering the grey region in the figure). Sampling a bad rock incurs a penalty of -10.

More formally, using the definitions from (Smith and Simmons 2004) we have the following.

- $S = \{(s_{\text{agent}}, s_{\text{rock}(1)}, \ldots, s_{\text{rock}(k)})\}$, where $s_{\text{agent}} \in \{(1,1), \ldots, (n,n)\}$ and $s_{\text{rock}(i)} \in \{\text{Good}, \text{Bad}\}, i = 1, \ldots, k$.
- $A = \{\text{North}, \text{West}, \text{South}, \text{East}, \text{Sample}, \text{Check}_1, \ldots, \text{Check}_k\}$ for the agent.
- $O = \{\text{Good}, \text{Bad}, \text{Null}\}$.
- $T(s_{\text{agent}}, a, s'_{\text{agent}})$ is deterministic, for $a \in A$.
- $Z(s'_{\text{agent}}, a, o) = \text{Ber}(\eta)$, where

$$\eta = \frac{1}{2}(1 + 2^{-d(s'_{\text{agent}}, s_{\text{rock}(a)})/\phi}),$$

  for $a \in \{\text{Check}_1, \ldots, \text{Check}_k\}$. The function $d(\cdot, \cdot)$ measures the distance (i.e., Euclidean distance) between the state of the agent and the rock intended to scan and ϕ is a constant parameter.
- $R(s_{\text{agent}}, a)$ is defined as
  - ▷ $R(s_{\text{agent}}, \text{Sample}) = +10$, if the rock is good
  - ▷ $R(s_{\text{agent}}, \text{Sample}) = -10$, if the rock is bad
  - ▷ $R(s_{\text{agent}}, a) = -1$, for $a \in \{\text{North}, \text{West}, \text{South}, \text{East}\}$
  - ▷ $R(s_{\text{agent}}, a) = 0$, for $a \in \{\text{Check}_1, \ldots, \text{Check}_k\}$

We tested our method on rock sample $(7, 8)$, $(20, 50)$, and $(20, 100)$, increasing the action space from 13 to 105. The positions of rocks in ***RockSample***$(7, 8)$ exactly follows code implemented in POMCP. While for ***RockSample***$(20, 50)$ and ***RockSample***$(20, 100)$, the placements of rocks are sampled uniformly at random once, prior to experiments. An example of the rock placement for these larger ***RockSample*** scenarios is in Figure 2(a) and (b), respectively.

The placements of rocks for ***RockSample***$(20, 50)$ are set at (16, 14), (5, 3), (10, 8), (8, 12), (6, 18), (16, 10), (19, 12), (12, 13), (3, 18), (2, 3), (11, 8), (6, 6), (5, 13), (11, 17), (3, 9), (13, 16), (1, 6), (0, 10), (5, 7), (1, 17), (18, 13), (16, 16), (7, 2), (3, 5), (8, 15), (8, 4), (14, 0), (8, 8), (19, 18), (18, 5), (19, 11), (6, 7), (5, 0), (17, 10), (4, 16), (2, 5), (10, 0), (18, 4), (8, 13), (4, 6), (1, 13), (18, 0), (12, 14), (7, 7), (13, 0), (15, 8), (6, 14), (13, 18), (4, 19), (19, 19).

The placements of rocks for ***RockSample***$(20, 100)$ are set at (8, 14), (11, 16), (18, 2), (2, 9), (0, 4), (8, 15), (11, 6), (18, 16), (5, 3), (10, 17), (15, 18), (6, 3), (1, 1), (8, 10), (0, 0), (13, 16), (2, 18), (3, 14), (10, 4), (12, 11), (7, 18), (12, 17), (16, 12), (14, 15), (7, 16), (11, 11), (4, 0), (14, 5), (6, 8), (1, 8), (6, 17), (6, 1), (0, 6), (3, 3), (17, 4), (13, 14), (17, 5), (5, 4), (17, 2), (4, 4), (19, 16), (8, 7), (4, 13), (17, 18), (7, 8), (10, 12), (14, 19), (16, 8), (7, 13), (1, 6), (4, 18), (15, 5), (18, 5), (11, 18), (18, 9), (11, 5), (19, 1), (15, 3), (3, 6), (10, 19), (12, 15), (17, 13), (12, 16), (19, 8), (2, 14), (5, 9), (9, 16), (2, 8), (4, 17), (3, 0), (13, 19), (6, 5), (15, 0), (10, 3), (4, 9), (13, 17), (0, 5), (11, 15), (19, 3), (7, 14), (11, 4), (18, 1), (5, 17), (16, 13), (3, 19), (17, 19), (5, 10), (16, 18), (16, 9), (3, 17), (19, 0), (5, 2), (15, 14), (16, 7), (9, 7), (18, 12), (2, 0), (2, 7), (17, 1), (0, 13).

B2. Navigation(d, n)

Figure 2(c) illustrates the navigation scenario $(d = 2, n = 30)$. An agent must navigate to a goal location (marked as red square) in a d-dimensional grid world populated by obstacles (marked as black regions), where each dimension is discretized into n cells. The agent initial position is not exactly known (the true position is the green square), but it must be at one of the 3^d cells on the upper left (marked by light blue squares). The agent can move to adjacent cells, within 3 cells away from its current position, resulting in 7^d possible actions. Its motion is accurate 90% of the time. The rest of the probability mass is divided equally among the $7^d - 1$ remaining cells. The agent can only observe the existence and position of walls surrounding its current cell, forming 2^{2d} observations. The observation function is perfect, though it is not sufficient to make the state to be fully

observable. The agent receives a $+1,000$ reward for reaching the terminal states and incurs a -1 penalty for every movement.

More formally, from the description above we have the following definitions.

- Define a grid world as $\mathcal{W} = [1, n]^d$, where n is the size and d is the dimension. $\mathcal{W}$ contains two parts: $\mathcal{W}_{\text{free}}$ and $\mathcal{W}_{\text{obstacle}}$. The way to generate $\mathcal{W}_{\text{free}}$ and $\mathcal{W}_{\text{obstacle}}$ follows four steps. For example in $d = 2$ navigation, we start with an empty gird world $\mathcal{W}$. Then
  1. add boundaries with thickness m_b,
  2. add obstacles with thickness m_o in the middle by: i) setting walls at $x = \{\lceil \frac{n+1}{2} \rceil - m_o + 1, \ldots, \lceil \frac{n+1}{2} \rceil + 1\}$ for any y; ii) setting walls at $y = \{\lceil \frac{n+1}{2} \rceil - m_o + 1, \ldots, \lceil \frac{n+1}{2} \rceil + 1\}$ for any x,
  3. delete obstacles with thickness m_f to get $\mathcal{W}_{\text{free}}$ if a cell $(x, y) \in \mathcal{W}$ is unwalkable, $\forall x, y \in \{\lceil \frac{n+1}{2} \rceil - m_f + 1, \ldots, \lceil \frac{n+1}{2} \rceil + \frac{m_f}{2} + 1\}$,
  4. set the goal cell in a corner.

  After setting the grid, we can obtain a grid world including walkable cells and unwalkable ones as well. For $d > 2$ problem, the way of constructing environment is the same.
- $S = \mathcal{W}_{\text{free}}$, where $m_b = 3, m_o = 2$ and $m_f = 6$.
- $A = \{(A_{\text{dim}_1}, \ldots, A_{\text{dim}_d})\}$, where each $A_{\text{dim}} = \{-3, -2, -1, 0, 1, 2, 3\}$ respect to the distance current state in this dimension.
- $O = \{(O_{\text{face}_1}, \ldots, O_{\text{face}_{2d}})\}$, where each $O_{\text{face}} = \{\text{Wall}, \text{NoWall}\}$.
- $T(s_{\text{agent}}, a, s'_{\text{agent}})$ has 90% accuracy of arriving at the desired s'_{agent} after executing action a, while arriving (wrongly) at other states with equal probability.
- $Z(s'_{\text{agent}}, a, o)$ is deterministic.
- $R(s_{\text{agent}}, a)$ is defined as
  - ▷ $R(s_{\text{agent}}, a) = +1,000$, if action a lead the agent reach pre-defined goal,
  - ▷ $R(s_{\text{agent}}, a) = -1$, otherwise.

B3. Hunting(n, u, v)

Figure 2(d) illustrates the scenario for $(11, 3, 3)$. Multiple (u) robots (green squares with letters) controlled by a centralized head, try to catch multiple (v) targets moving in a grid-world of size $n \times n$ populated by obstacles (black regions). At the beginning, the targets' positions are not known, and represented as uniform distributions over the free cells (colored pink). The true positions of the targets (which are unknown) are marked by red squares, The state space is the Cartesian product of the positions of the robots and the targets, while the action and observation spaces are the Cartesian products of all of the robots' actions and observations. At each step, each robot can stay where it is, move to one of the 8 cells adjacent to its current position, or catch a target. Their motion has no error. Furthermore, at each step, each robot can perfectly detect whether there is target(s) located in the same cell as itself or in one cell to its North, South, East, or West directions. Note that although its detection is perfect, a robot cannot distinguish which target is being detected nor the exact position (out of the five cells) of the target. A small penalty -1 is imposed on movement action for each robot. The 'catch' action yields a $+100$ reward if the agent is in the same cell as the target(s), otherwise the action incurs a penalty of -100. The targets know exactly the positions of the robots and always move to the direction farthest from the closest robot.

More formally, from the description above we have the following definitions.

- $\mathcal{W}$ is shown in Figure 2(d). It contains the forbidden area $\mathcal{W}_{\text{obstacle}}$ (marked in black) and walkable area $\mathcal{W}_{\text{free}} = \mathcal{W} \setminus \mathcal{W}_{\text{obstacle}}$.
- $S = \{\big(s_{\text{agent}_1}, \ldots, s_{\text{agent}_u}, s_{\text{target}_1}, \ldots, s_{\text{target}_v}\big)\}$, where $s_{\text{agent}_i} \in \mathcal{W}_{\text{free}}, i = 1, \ldots, u$ and $s_{\text{target}_j} \in \mathcal{W}_{\text{free}}$, $j = 1, \ldots, v$.
- $A = \{\big(A_{\text{agent}_1}, \ldots, A_{\text{agent}_u}\big)\}$, where each $A_{\text{agent}_i} = \{\text{Stay}, \text{North}, \text{Northwest}, \text{West}, \text{Southwest}, \text{South}, \text{Southeast}, \text{East}, \text{Northeast}, \text{Catch}\}, i = 1, \ldots, u$.
- $O = \{(O_{\text{agent}_1}, \ldots, O_{\text{agent}_u})\}$, where each $O_{\text{agent}} = \{\text{Yes}, \text{No}\}$.
- $T(s_{\text{agent}}, a, s'_{\text{agent}})$ is deterministic. $T(s_{\text{target}}, a, s'_{\text{target}})$ in *Hunting-smart* follows the strategy in *Tag* (Pineau, Gordon, and Thrun 2003). That is, whenever a robot and a target occupy the same cell, the target can still get away, unless the robot performs the action 'catch'. While in *Hunting-normal*, once a robot and a target are in the same cell, the target cannot escape.
- $Z(s'_{\text{agent}}, a, o)$ has the deterministic effect of detecting targets around the agent
- $R(s_{\text{agent}}, a)$ is defined as
  - ▷ $R(s_{\text{agent}}, \text{Catch}) = +100$, if the target is tagged correctly,
  - ▷ $R(s_{\text{agent}}, \text{Catch}) = -100$, if there is no target,
  - ▷ $R(s_{\text{agent}}, a) = -1$, otherwise.