PUMA: Planning under Uncertainty with Macro-Actions

Ruijie He
ruijie@csail.mit.edu
Massachusetts Institute of Technology
Cambridge, MA 02139 USA

Emma Brunskill
emma@cs.berkeley.edu
University of California, Berkeley
Berkeley, CA 94709 USA

Nicholas Roy
nickroy@csail.mit.edu
Massachusetts Institute of Technology
Cambridge, MA 02139 USA

Abstract

Planning in large, partially observable domains is challenging, especially when a long-horizon lookahead is necessary to obtain a good policy. Traditional POMDP planners that plan a different potential action for each future observation can be prohibitively expensive when planning many steps ahead. An efficient solution for planning far into the future in fully observable domains is to use temporally-extended sequences of actions, or "macro-actions." In this paper, we present a POMDP algorithm for *planning under uncertainty with macro-actions* (PUMA) that automatically constructs and evaluates open-loop macro-actions within forward-search planning, where the planner branches on observations only at the end of each macro-action. Additionally, we show how to incrementally refine the plan over time, resulting in an anytime algorithm that provably converges to an ϵ-optimal policy. In experiments on several large POMDP problems which require a long horizon lookahead, PUMA outperforms existing state-of-the art solvers.

Most partially observable Markov decision process (POMDP) planners select actions conditioned on the prior observation at each timestep: we refer to such planners as *fully-conditional*. When good performance relies on considering different possible observations far into the future, both online and offline fully-conditional planners typically struggle. An extreme alternative is *unconditional* (or "open-loop") planning where a sequence of actions is fixed and does not depend on the observations that will be received during execution. While open-loop planning can be extremely fast and perform surprisingly well in certain domains[1], acting well in most real-world domains requires plans where at least some action choices are conditional on the obtained observations.

This paper focuses on the significant subset of POMDP domains, including scientific exploration, target surveillance, and chronic care management, where it is possible to act well by planning using conditional sequences of open-loop, fixed-length action chains, or "macro-actions." We call this approach *semi-conditional* planning, in that actions are chosen based on the received observations only at the end of each macro-action.

[1]For a discussion of using open-loop planning for multi-robot tag for open-loop planning see Yu et al. (2005).

We demonstrate that for certain domains, planning with macro-actions can offer performance close to fully-conditional planning at a dramatically reduced computational cost. In comparison to prior macro-action work, where a domain expert often hand-coded a good set of macro-actions for each problem, we present a technique for automatically constructing finite-length open-loop macro-actions. Our approach uses sub-goal states based on immediate reward and potential information gain. We then describe how to incrementally refine an initial macro-action plan by incorporating successively shorter macro-actions. We combine these two contributions in a forward-search algorithm for *planning under uncertainty with macro-actions* (PUMA). PUMA is an anytime algorithm which guarantees eventual convergence to an ϵ-optimal policy, even in domains that may require close to fully-conditional plans.

PUMA outperforms a state-of-the-art POMDP planner both in terms of plan quality and computational cost on two large simulated POMDP problems. However, semi-conditional planning does not yield an advantage in all domains, and we provide preliminary experimental analysis towards determining a priori when planning in a semi-conditional manner will be helpful. However, even in domains that are not well suited to semi-conditional planning, our anytime improvement allows PUMA to still eventually compute a good policy, suggesting that PUMA may be viable as a generic planner for large POMDP problems.

POMDPs

A partially observable Markov decision process is the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{Z}, p(s'|s,a), p(z'|s',a), r(s,a), b_0, \gamma \rangle$, where $\mathcal{S}$ is a discrete set of states, $\mathcal{A}$ is a discrete set of actions, $\mathcal{Z}$ is a discrete set of observations, $p(s'|s,a)$ is the probability of transitioning to from state s to s' after taking action a, $p(z'|s',a)$ is the probability of receiving observation z after taking action a and transitioning to state s', $r(s,a)$ is the reward for state s and action a, b_0 is an initial probability distribution over states, and γ is a discount factor.

The past history of actions and observations can be summarized by a sufficient statistic known as a belief state b_t. The belief state can be updated after each action taken, and observation received, using a Bayes filter,

$$b_{t+1}(s') \propto p(z|a,s') \sum_{s \in \mathcal{S}} p(s'|s,a) b_t(s). \quad (1)$$

The goal of POMDP planning is to compute a policy $\pi : b \rightarrow a$ which map beliefs to actions, with the highest value function $V^{\pi}(b)$. $V^{\pi}(b)$ specifies the expected infinite-horizon reward of executing policy π starting at any belief b,

$$V^{\pi}(b)=\sum_{s\in\mathcal{S}} r(s,\pi(b))b(s)+\gamma\sum_{z\in Z} p(z|b,\pi(b))V^{\pi}(b^{\pi(b),z}). \quad (2)$$

Forward Search POMDP Planning

Current state-of-the-art, fully-conditional, offline planners, such as the SARSOP algorithm (Kurniawati, Hsu, and Lee 2008), compute an explicit representation of the value function. In contrast, online forward search planners (see the survey by Ross, Pineau, Paquet and Chaib-draa, 2008) estimate the belief-action value only for the current belief. The value of each action is computed by constructing a forward search tree that alternately expands potential actions, and branches on possible observations, to a fixed horizon H.

Unfortunately, the computational complexity of fully-conditional forward search planning scales exponentially with the horizon length, due to the breadth-first expansion of possible actions and observations. Particularly for large problems, the growth in cost with search depth typically limits a full forward search to a short horizon, causing poor performance in problems that require a long lookahead. To improve the performance of short horizon search, domain-specific heuristics can be used to approximate the value at the leaves, but human intervention is required to choose the heuristics. An alternative to heuristics is to compute an offline approximation of the value function to be used at the tree leaves. Unfortunately it is often difficult to find a small enough reduction of the original problem that can be solved and yet still provides a good value estimate at the leaves of the search tree.

By using macro-actions to restrict the policy space and reduce the action branching factor, semi-conditional forward-search POMDP planners can reduce the computational cost. If $\tilde{\mathcal{A}}$ is the set of macro-actions and L is an upper-bound on the length of each macro-action, the cost due to action branching shrinks from $O(|\mathcal{A}|^H)$ to $O(|\tilde{\mathcal{A}}|^{H/L})$. This reduction in computational complexity allows the planner to search much farther into the future, which may enable good plans to be found without relying on a heuristic at the leaves. Implicit in this statement is the assumption that only a subset of the potential macro-actions between lengths 1 to L will be considered, otherwise the computational savings will be lost. However, this choice also directly restricts the plan space of the search process, possibly resulting in reduced performance. Therefore, it is crucial to be able to generate macro-actions that are anticipated to be part of a good plan. Prior work achieved this by hand-coding macro-actions (such as work by He and Roy, 2009). In this paper we present an automatic way to generate finite-length, open-loop macro-actions that empirically produces good performance.

Generating Macro-actions

The best macro-action that can be executed from any two beliefs often varies greatly, especially since the two beliefs could have support over very different regions of the state space. Rather than using a large, fixed set of macro-actions for every belief, our planner instead re-generates a new set of macro-actions at every step, and evaluates the values of these macro-actions using forward search.

To generate macro-actions automatically, a naïve approach would be to chain together randomly sampled primitive actions. However, the space of possible macro-actions of maximum length L grows exponentially larger with L, making it unlikely that a randomly-generated macro-action would be the best possible open-loop action sequence that the agent could have executed. In contrast, many macro-action planners in fully-observable domains create mini-policies designed to achieve sub-goals (for example, see McGovern 1998, and Stolle and Precup 2002). We hypothesize that sub-goals are similarly applicable within a POMDP context.

Reward exploitation/information gathering weights

To select suitable sub-goals, we take inspiration from Cassandra et al.'s 1996 and Hsiao et al.'s 2008 work showing that good POMDP performance can be achieved by choosing either actions with high rewards or actions with a large gain in information under the current belief. Our macro-actions are therefore finite-length open-loop plans designed to take the agent from a possible state under the current belief towards a state that will provide either a large immediate reward or a large gain in information.

We compute two set of weights, w_r and w_i, that correspond to the reward exploitation and information gain metric, respectively. Two weights, $w_r(s)$ and $w_i(s)$ are computed for each state: these weights specify the probability that the planner will choose state s as a sub-goal for generating a reward-exploitation or information-gain macro-action. To calculate the reward exploitation weights, we first compute a reward exploitation heuristic (RE) for each state s:

$$RE(s) = \max_{a\in\mathcal{A}}(R_S(s,a) - R_{min})/(R_{max} - R_{min}). \quad (3)$$

$RE(s)$ is the best immediate reward for state s, normalized by the minimum and maximum instantaneous rewards. This ensures that RE is strictly positive. The reward exploitation weights are computed by normalizing the heuristic values:

$$w_r(s) = RE(s)/\sum_s RE(s). \quad (4)$$

To compute w_i, we first compute an information gain heuristic (IG) which computes the entropy associated with the observation emission probabilities from each state s, versus a uniform distribution:

$$IG(s) = \sum_{k=1}^{|\mathcal{Z}|}\left(-\frac{1}{|\mathcal{Z}|}\log(\frac{1}{|\mathcal{Z}|}) + p(z_k|s)\log(p(z_k|s))\right).$$

The lower the entropy over the observation emission probabilities, the higher the probability that the agent will obtain an observation that provides positive information about the state of the world. To compute the information gain weights we normalize the IG values to sum to one:

$$w_i(s) = IG(s)/\sum_s IG(s). \quad (5)$$

As both w_r and w_i are belief-independent, the sets of weights only need to be computed once for each problem.

Algorithm 1 GENERATEMACROS()

Require: Current belief b_c, Num macros M, Max length L

for $k = 1$ to M **do**
  Sample state $s_{s,k}$ from b_c
  $V_g \leftarrow 0$
  while $V_g(s_{s,k}) == 0$ **do**
    Sample sub-goal state $s_{g,k}$
    Compute "goal-oriented" MDP g for sub-goal $s_{g,k}$
  end while
  $j \leftarrow 1$, Set state $s_c \leftarrow s_{s,k}$
  while $j < L$ and state $s_c \neq$ goal state s_g **do**
    $\tilde{a}_k(j) = \pi_g(s_c)$ (select action using MDP g policy)
    $s_c = \text{argmax}_{s'} P(s'|s_c, \tilde{a}_k(j))$
    $j \leftarrow j + 1$
  end while
  Add macro-action $\tilde{a}_k$ to $\tilde{\mathcal{A}}$
end for
Return macro-action set $\tilde{\mathcal{A}}$

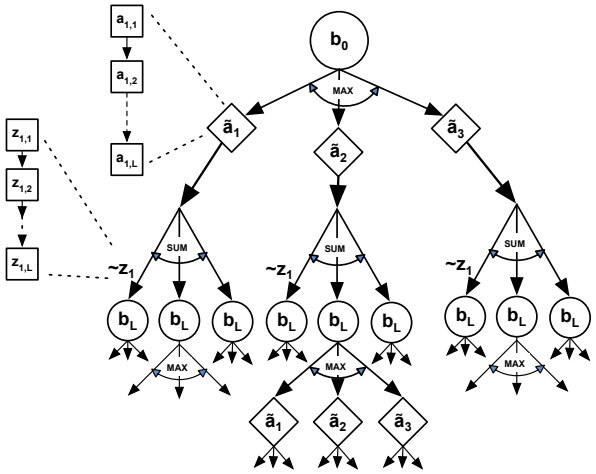


Figure 1: Macro Action forward search tree

Goal-oriented MDP

We will shortly describe how we use the computed weights to sample a state $s_{g,k}$. Given a sampled $s_{g,k}$, we create a "goal-oriented" MDP where $s_{g,k}$ has a high reward, all other states have zero rewards, and the transition model is identical to the POMDP transition model. A discount factor $0 < \gamma < 1$ biases the policy towards action sequences that will arrive at the goal state in the smallest number of timesteps. The resultant MDP policy therefore provides a mapping from states to actions, which maximizes a value function that balances between the probability of reaching the goal state and reaching that state in the shortest time possible.

Finally, bringing all the pieces together, to generate a macro-action from the current belief state b_c, we first sample a start state $s_{s,k}$ from b_c. We then randomly choose whether to use the reward exploitation weights w_r or information gain weights w_i, and then sample a sub-goal state $s_{g,k}$ using the probabilities specified by the chosen weights. We next check that under the goal-oriented MDP for $s_{g,k}$, that the sampled start state $s_{s,k}$ can reach the goal state $s_{g,k}$: this is simple to verify as if $s_{s,k}$ cannot reach the goal, it will have a value of zero in the goal-oriented MDP. Assuming the sample goal can be reached from $s_{s,k}$, we generate a macro-action by simulating execution of the goal-oriented MDP, assuming that the maximum likelihood state is reached at every step. Actions are added until either the sub-goal $s_{g,k}$ is reached or the macro-action is of maximum length L. We repeat the macro-action generation process M times for a particular belief to generate M unique macro-actions. Algorithm 1 summarizes our macro-action generation process.

Forward Search Using Macro-Actions

We now present an initial version of planning under uncertainty using macro-actions via online forward-search. Our algorithm builds upon a generic online forward search for POMDPs (Ross et al. 2008), alternating between a planning and execution phase at every iteration. During the planning phase, the best macro-action $\tilde{a}_{opt}$ is chosen and the first action a_t of this action sequence is executed. The agent then receives an observation, updates the current belief using Equation 1, and the cycle repeats.

To select an action for the current belief, we first generate a set of M macro-actions with maximum length L from the current belief b_c using GENMACROS described in the previous section (Algorithm 1). We set M to always be at least as large as the number of primitive actions $|\mathcal{A}|$.

The forward-search tree is then constructed by considering the potential future action and observation trajectories from the initial belief state b_c. The best macro-action $\tilde{a}_{opt}$ is the macro-action that has the highest expected future reward. To evaluate the expected reward of a macro-action, note that a macro-action $\tilde{a}$ of length L will generate a sequence of L beliefs $[b_0, \ldots, b_L]$ when executed. Each belief will have an expected immediate reward $\tilde{r}_k$ where $\tilde{r}_k = \sum_s r(s, a_k) b_k(s)$. The sequence of beliefs for a macro-action is a function of the observations received during the execution of $\tilde{a}$. As a result, the value of a macro-action is the expected value taken over all possible belief sequences, according to the probability of each belief sequence given by the corresponding observation sequence. For each posterior belief b_L that results from each different observation sequence, we recursively compute the value of the best macro-action from *that* posterior belief.

Therefore, to compute the expected reward for a macro-action $\tilde{a}$ from the root, we simulate its execution and generate all possible observation sequences from the root belief, resulting in a set of possible expected rewards and posterior beliefs. Posterior beliefs are sequentially updated according to Equation 1. For each posterior belief, we generate a new set of macro-actions and again expand out the forward search, as shown in Figure 1. We repeat this process until the tree depth reaches a pre-specified horizon H. The expected values of the beliefs are then carried back up the tree, taking the maximum expected value over the macro-actions at each belief, and the sum over observation sequences.

In practice, the number of possible observation sequences that could occur during a single macro-action scales exponentially with the macro-action length L, for a total of $O(|\mathcal{Z}|^L)$ sequences. To reduce the computational cost of expanding all possible observation trajectories, when con-

Algorithm 2 PUMA

Require: Current belief b_c, Planning horizon H, Planning time t_{plan}, Num macros M, Num obs. traj. samples N_z

$t_{elapsed} = 0$; $L = H$; $b_e = b_c$
while $t_{elapsed} < t_{plan}$ **do**
  $\tilde{\mathcal{A}}$ = GENMACROS(b_e, M, L)
  for each macro-action $\tilde{a}_k \in \tilde{\mathcal{A}}$ **do**
    for $i = 1 : N_z$ **do**
      Sample observation traj $\tilde{z}_i$ from b_e and $\tilde{a}_k$
      b_i =UPDATEBELIEF($b_e, \tilde{a}_k, \tilde{z}_i$)
      Compute $V(b_i)$ by recursive forward search
    end for
    $Q(b_e, \tilde{a}_k) = \sum_{i=1}^{N_z} V(b_i)/N_z$
  end for
  Choose $\tilde{a}_{refine}$ and b_{refine} to refine
  $L = Length(\tilde{a}_{refine})/2$; $b_e = b_{refine}$
end while
Return $\tilde{a}_{opt} = \text{argmax}_{\tilde{a}}\, Q(b_c, \tilde{a})$

structing a forward search tree, we only sample a small number N_z of observation sequences for each macro-action, which significantly reduces the computational complexity.

When the search reaches a depth of H primitive actions (H/L macro-actions), the value of the posterior belief nodes is set to zero. As mentioned previously, a common practice is to use value heuristics at leaf nodes, but by being able to search deeper efficiently, our macro-action planners instead enable us to achieve good performance on several longer horizon problems without using value heuristics to estimate the cost-to-go from the leaf belief nodes.

Anytime Iterative Refinement

One limitation of our procedure for generating macro-actions is that plans based only on open-loop action sequences between sub-goals may not correspond to the optimal, or even a good policy. In particular, so far we have assumed that the user can provide a reasonable estimate of the maximum macro-action length L. However, it may not be obvious how to set L for a given problem; in some cases where fully-conditional planning is required for good performance, every primitive action should depend on the previous observation and L should be 1. To handle this issue, we extend the planner to allow the algorithm to converge to ϵ-optimal performance for any POMDP in an any-time manner. Algorithm 2 shows the full PUMA algorithm.

If additional computation time is available after the initial tree is built and evaluated, we iteratively refine macro-action branches in the forward-search tree. To select the next macro-action and belief node to refine, we find the macro-action that is nearest to the root belief node and is of length $L > 1$. Ties are broken arbitrarily. Once a macro-action $\tilde{a}_{refine}$ and belief node b_{refine} is chosen, the sequence of actions in the macro-action is pruned from its original length ($Length(\tilde{a}_{refine})$) to a maximum length of $Length(\tilde{a}_{refine})/2$. A new belief node b' is created by taking the pruned macro-action from b_{refine}, and $V(b')$ is computed using forward search. The new value is propagated up the tree, and if the root belief node b_c has a macro-action with a higher expected value, the best macro-action $\tilde{a}_{opt}$ is updated. Note that we always include the value of the old L-length $\tilde{a}_{refine}$ when computing the new best macro-action, therefore ensuring that the new policy space after a refinement has occurred always includes the old policy space. This cycle repeats until all computation time available for planning t_{plan} has expired; this process gradually increases the amount of conditional branching in the tree.

The process of generating macro-actions, performing an initial search and then iterative improvement is our complete *planning under uncertainty with macro-actions* algorithm, PUMA.

Analysis

For completeness we include the following performance bound:

Theorem 1. *Let H be the primitive-action horizon length used in the PUMA algorithm. Assume that at least $|\mathcal{Z}|$ unique observation sequences are sampled per macro-action. Then, as the amount of computational time available for selecting an action increases, the PUMA policy followed will converge to an ϵ-optimal policy where*

$$\epsilon = \frac{\gamma^{H+1} \max_{a,s} r(s,a)}{1-\gamma}. \tag{6}$$

Proof. First note the PUMA macro-action creation process ensures that at a given macro-action branching point, there are always at least $|\mathcal{A}|$ unique macro-actions. As the amount of time available to select an action for the root belief increases, the PUMA splitting criteria ensures that eventually all macro-actions in the forward search tree will be refined to length one. Combined with the prior condition on the number of unique macro-actions and the stated assumption on the number of unique sampled observation sequences, the forward search tree will become a fully-conditional H-depth tree, with all actions are expanded at each depth, followed by all observations. Therefore, the action-values at the root belief will be exactly the H-step value of taking those actions from the root belief. Now, recall from Puterman (1994) that the difference between the belief-action H-horizon optimal value $Q_H(b,a)$, and the belief-action infinite-horizon optimal value $Q^*(b,a)$ is bounded:

$$Q^*(b,a) - Q_H(b,a) \leq \frac{\gamma^{H+1} \max_{a,s} r(s,a)}{1-\gamma} \tag{7}$$

Therefore, since in the limit of computational time PUMA converges to the optimal H-horizon policy for the current belief, the value of the PUMA policy is guaranteed to be at worst ϵ-close to the optimal infinite-horizon policy. □

Note Theorem 1 provides a principled way to select the planning horizon H based on the desired ϵ value. In addition, Theorem 1 shows that PUMA removes a limitation of many macro-action planners that do not guarantee good performance on problems that require a fully-conditional policy.

Performance and Planning Time

If all observation trajectories are exhaustively enumerated and weighted by their probability, then the computed root

belief-action values are the exact belief-action values of following the macro-action tree policy. Since the new policy space after a macro-action refinement always includes the original policy space, plus new potential policies due to the additional conditional branching, the new (exact) computed values at the root node will be at least the same value as they were before the split occurred, or a higher value. In this case, the computed root-belief node value is guaranteed to monotonically increase as the tree is iteratively refined given more planning time.

The computational complexity of PUMA is a function of the number of nodes in the forward search tree, plus an additional cost for macro-action generation. MDPs with different sampled sub-goals can be solved once and the solutions stored, amortizing the cost over multiple steps of planning. As we iteratively refine each macro-action, the number of nodes grows to a full forward search. Therefore the PUMA cost converges to the cost of a full forward search, plus the overhead of iterative macro-action refinement.

Due to the prohibitive computational cost of enumerating all observation sequences in large horizons, PUMA only samples a small number of observation trajectories per macro-action. This yielded good empirical performance.

Experiments

The PUMA algorithm consists of two phases: an initial search process using automatically generated macro-actions, and then a subsequent phase of any-time iterative improvement. In order to analyze these two phases, we first evaluated PUMA with no iterative improvement, and then examined the effect of the refinement process.

Experiments: Macro-Action Generation We first compared the PUMA algorithm without iterative improvement to SARSOP (Kurniawati, Hsu, and Lee 2008), a state-of-the-art offline POMDP planner. We begin with a variant of the scientific exploration RockSample (Smith and Simmons 2004) problem called the Information Search Rocksample (ISRS) problem (He and Roy 2009), shown in Figure 2(a). In ISRS(n,k) an agent explores and samples k rocks in a $n \times n$ grid world. The positions of the agent (pink square) and the rocks (circles) are fully observable, but the value of each rock (good or bad) is unknown to the agent. At every timestep, the agent receives a binary observation of the value of each rock. The accuracy of this observation depends on the agent's proximity to rock information beacons (yellow squares) that each correspond to a particular rock. Unlike the original Rocksample problem, good performance on ISRS requires searching far into the future, in that an agent must plan first to visit beacons to identify the value of rocks before sampling rocks. Information is physically separated from reward.

The number of sampled macro-actions M is an input parameter: as we increase M PUMA quickly achieves performance equivalent to a macro-action algorithm that uses the hand-coded macros (Figure 2(b)) at a slightly larger computational cost. This suggests our macro-action generation algorithm is a good heuristic for finding effective macro-actions. H was limited to 15 for computational reasons. L was set to 5, based on experimental evidence that forward-search action branching is normally computationally limited to 3 successive branches ($H/3 = 5$).

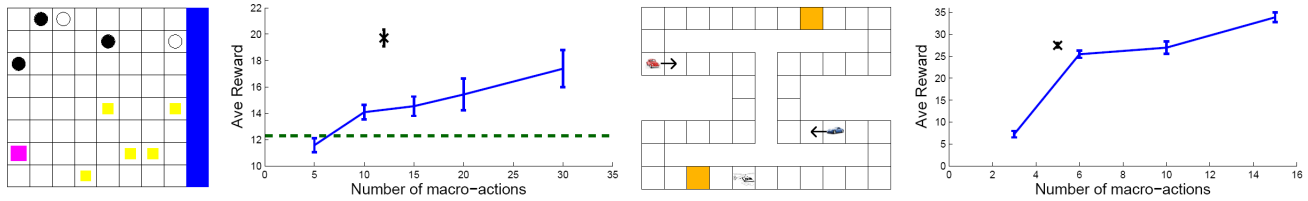
	Ave. reward	Offline time	Online time
Noisy Tiger			
SARSOP	**-38.61** $\pm$ 0.53	0.50	0.000
Naive fs	-41.19 $\pm$ 1.70	0.000	0.018
PUMA	-41.67 $\pm$ 1.66	0.000	5.18
ISRS (8,5)			
SARSOP	12.10 $\pm$ 0.26	10000	0.000
Naive fs	9.56 $\pm$ 1.08	0.000	3.36
Hand-coded	**19.71** $\pm$ 0.63	0.000	0.74
PUMA	17.38 $\pm$ 1.41	0.000	162.48
Tracking (50)			
Naive fs	19.58 $\pm$ 0.42	0.000	0.023
Hand-coded	27.48 $\pm$ 0.49	0.000	1.010
PUMA	**35.55** $\pm$ 1.28	0.000	28.52

Table 1: Summary of experimental results using the best explored parameters for each algorithm.

As a second example, we introduce a larger, multi-target extension of the single target-tracking problem introduced by Hsu, Lee, and Rong (2008): see Figure 2(c). At every timestep, each of the two targets move probabilistically zero, one or two steps forward on a track with 50 states. The agent can move along the track one or two steps forward, one or two steps backwards, or can stay at the same spot. The agent receives a large positive reward if it is in the vicinity of the targets whenever either of the targets land in the yellow square regions, and a negative reward if it is not close to a region when a target enters a yellow region. A small cost is incurred for every movement action. The agent obtains a noisy observation of the target's position when it is in the vicinity of the target.

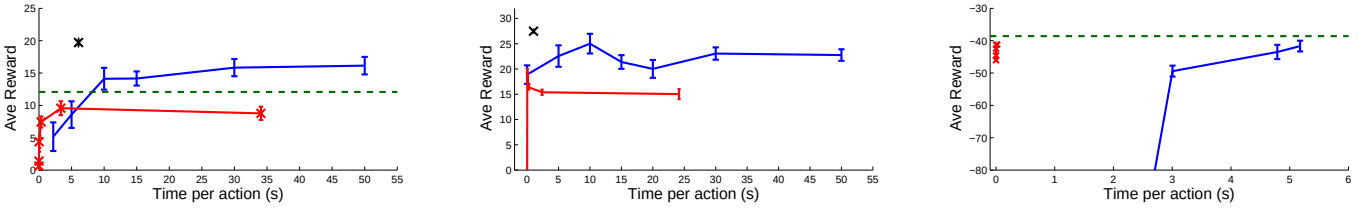
Although this problem is inherently factored, offline approaches are typically unable to take advantage of the factored properties of the problem domains. The problem has 50 environmental states, but the model results in $50^3 = 125,000$ non-factored states, a problem that is not representable, let alone solvable, by offline non-factored approaches. Note that the forward search approach allows us to exploit the factored nature of the problem domain for increased efficiency when updating the beliefs and calculating expected rewards. We compared PUMA to a traditional forward search, as well as a macro-action forward search algorithm where the macro-actions were specified using domain knowledge. Figure 2(d) shows that when a large number of macro-actions are automatically generated, the PUMA algorithm actually outperforms a policy using hand-coded macro-actions.

Experimental results: Refinement We examined the performance of iterative refinement on ISRS and tracking. As the amount of available online computation time increases, the iterative refinement of the macro-action forward search tree improves performance in the ISRS problem, as shown in Figure 3(a). Using 15 macro-actions at each branch point, the algorithm evolves from being an unconditional planner ($L = H$), where performance is worse than both a traditional forward search and SARSOP, into a semi-conditional planner where conditioning at different points in the planning horizon results in better performance than alternative POMDP approaches. Iterative refinement also



(a) ISRS (8,5) (b) PUMA performance (ISRS) (c) Target-tracking (50) (d) PUMA performance (Tracking)

Figure 2: (a) ISRS problem: Agent (purple square) has to retrieve rocks of high value (white circles). It receives noisy observations of each rock's value; observation accuracy increases with proximity to information beacons (yellow square). (c) Tracking: Yellow squares are regions of high value. When the agent (helicopter) is over a target (cars) as the target enters a yellow square, a large positive reward is obtained. If a target enters a yellow square without the agent present, a large negative reward is gained. (b,d) Performance of PUMA (blue line) without refinement improves when more macro-actions are used, outperforming SARSOP (green dotted line) and for target-tracking, even outperforming a forward search algorithm which uses hand-coded macro-actions (black cross).



(a) ISRS (8,5), PUMA 15 macro-actions (b) Target-tracking (50), PUMA 5 macro-actions (c) Noisy Tiger, PUMA 3 macro-actions

Figure 3: Performance of PUMA (blue line) as more planning time is allocated for iterative refinement, compared to traditional forward search (red line) out to a progressively longer horizon, SARSOP (green dotted line), and hand-coded macro-action forward search (black cross).

improves performance in the target-tracking problem (Figure 3(b)), though empirically the performance does not improve monotonically with more planning time.

We also applied PUMA to a POMDP benchmark problem, Tiger, to demonstrate that the design of PUMA still allows good performance on problems that are not well suited to macro-actions, albeit at higher computational cost. Tiger is a small POMDP problem that offers two challenges to semi-conditional planners. First, it requires fully conditional planning to achieve good performance. Second, good action sequences depend on multiple "listen" actions that do not actually change the state itself: approaches such as the Milestone Guided Sampling (MiGS) algorithm of Kurniawati et al. (2009) that generate macro-actions based only on sampling state sub-goals will fail. To require planning to a longer horizon, the observation noise was increased from the standard model of 0.15 noise to 0.35 noise, such that the agent has a lower probability of obtaining an accurate observation when it executes the "listen" action: we call this "Noisy Tiger." PUMA iteratively refines the macro-actions until it reaches a primitive forward search of horizon 5, where all primitive actions are considered at each depth: Figure 3(c) shows the PUMA algorithm converging towards the optimal policy as more computation time is made available for planning. Table 1 summarizes the results of the different algorithms for each POMDP problem.

Experiments: When to use semi-conditional planners? To deploy semi-conditional planners effectively, it is necessary to understand the conditions under which a planner can generate good policies without having to condition on the observations after every action. Ideally one would have a classifier function or score that, given an input POMDP specification, could determine whether semi-conditional planning is likely to be beneficial. Towards this goal, our preliminary investigations suggest that reward variance is likely to be an important domain characteristic which helps determines whether frequent conditioning on observations is required. More precisely, if $R_\mu(a_i)$ and $R_\sigma(a_i)$ are the reward mean and variance, respectively, for action a_i over all states, the average reward variance $R_{\bar{\sigma}}$ can be found by averaging $R_\sigma(a_i)$ across all actions. A small $R_{\bar{\sigma}}$ means that taking a particular action from any of the states will result in very similar rewards, except perhaps for a few states where the agent will receive a drastically different reward for taking the same action. Therefore, if $R_{\bar{\sigma}}$ is small, a planner can often accurately calculate the expected reward associated with a macro-action without having to condition after every action.

We explored this hypothesis by first modifying the ISRS problem so that the agent's position is partially observable and its transition model is stochastic. We also introduce "potholes" into the problem, whereby the agent incurs a negative reward when it enters any grid cell with a pothole. Increasing the number of potholes also increases the average reward variance $R_{\bar{\sigma}}$. Figure 4 shows the performance of PUMA, versus SARSOP, as the reward variance increases. As the reward variance increases, the semi-conditional planner experiences a much steeper decline in performance relative to SARSOP, as our hypothesis predicted. Nevertheless, measuring the reward variance alone ignores the observation model and process noise, both of which affect the planner's ability to accurately plan for many steps in an open-loop fashion. We seek to explore other criteria in future research.

Related Work

Macro-action planning formalisms for fully-observable environments include semi-Markov Decision Processes and the options framework (Sutton, Precup, and Singh 1999).

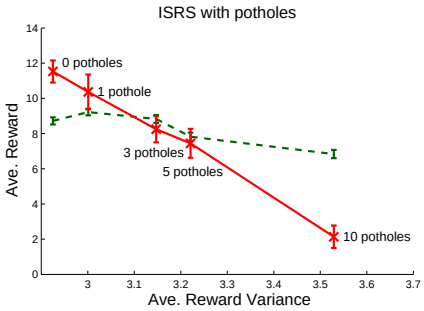


Figure 4: Performance of PUMA (red line) relative to SARSOP (green dotted line) vs. reward variance.

Prior work on macro-action generation includes the peak-to-peak heuristic of Iba (1989) and the sub-goal method of McGovern and Barto (2001).

For planning in partially observable MDPs, Pineau, Gordon and Thrun (2003) used a hierarchical approach for a robotic task. Other POMDP macro-action robotics planners by Theocharous and Kaelbling (2003), and Hsiao, Lozano-Pérez and Kaelbling (2008) provide good empirical performance, but do not focus on learning the macro-actions or on providing performance guarantees. Recently Toussaint, Charlin and Poupart (2008) presented an exciting inference approach for automatically learning a hierarchical finite state controller POMDP planner using expectation maximization, though their experimental results struggled with some large POMDP benchmark problems.

The closest related approach to PUMA is the MiGS algorithm by Kurniawati et al. (2009). There are three key differences between our PUMA approach and MiGS. First, iterative refinement enables PUMA to produce good solutions for generic POMDPs, including domains like Tiger which require sensing-only action sequences. The current implementation of MiGS will struggle because it only constructs macros by sampling states, though it should also be possible for MiGS to be extended to include an iterative refinement component. Second, PUMA's forward search procedure will scale better in large spaces since MiGS is an offline approach that uses a value function backup operator that scales as $O(|\mathcal{S}|^2|\mathcal{Z}||\mathcal{A}|)$. Finally, PUMA can exploit factored models to quickly compute belief updates and expected reward calculations for large state problems, but MiGS' value function representation cannot directly leverage factored structure.

Conclusion

In summary, we have presented an anytime semi-conditional forward-search planner for partially observable domains. PUMA automatically constructs macro-actions, and iteratively refines the macro-actions as planning time allows. PUMA achieves promising experimental results relative to state-of-the-art offline planners and a similar planner that uses hand-selected macro-actions, on several large domains requiring far-lookahead, and can also achieve good performance on domains that require fully-conditional planning.

Acknowledgements

We wish to thank the anonymous reviewers for their feedback. We would also like to thank Finale Doshi-Velez, Alborz Geramifard, Josh Joseph, and Javier Velez for valuable discussions and comments.

References

Cassandra, A.; Kaelbling, L.; and Kurien, J. 1996. Acting under uncertainty: Discrete Bayesian models for mobile robot navigation. In *IROS*.

He, R., and Roy, N. 2009. Efficient POMDP Forward Search by Predicting the Posterior Belief Distribution. Technical report, MIT.

Hsiao, K.; Lozano-Pérez, T.; and Kaelbling, L. 2008. Robust belief-based execution of manipulation programs. In *WAFR*.

Hsu, D.; Lee, W.; and Rong, N. 2008. A point-based POMDP planner for target tracking. In *ICRA*.

Iba, G. 1989. A heuristic approach to the discovery of macro-operators. *Machine Learning* 3(4):285–317.

Kurniawati, H.; Du, Y.; Hsu, D.; and Lee, W. 2009. Motion Planning under Uncertainty for Robotic Tasks with Long Time Horizons. In *ISRR*.

Kurniawati, H.; Hsu, D.; and Lee, W. 2008. SARSOP: Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces. In *RSS*.

McGovern, A., and Barto, A. 2001. Automatic discovery of subgoals in reinforcement learning using diverse density. In *ICML*.

McGovern, A. 1998. acQuire-macros: An algorithm for automatically learning macro-actions. In *NIPS AHRL*.

Pineau, J.; Gordon, G.; and Thrun, S. 2003. Policy-contingent abstraction for robust robot control. In *UAI*.

Puterman, M. 1994. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, Inc. New York, NY, USA.

Ross, S.; Pineau, J.; Paquet, S.; and Chaib-draa, B. 2008. Online planning algorithms for POMDPs. *JAIR* 32:663–704.

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

Stolle, M., and Precup, D. 2002. Learning options in reinforcement learning. *Lecture Notes in Computer Science*.

Sutton, R.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence* 112(1):181–211.

Theocharous, G., and Kaelbling, L. 2003. Approximate planning in POMDPs with macro-actions. *NIPS*.

Toussaint, M.; Charlin, L.; and Poupart, P. 2008. Hierarchical POMDP controller optimization by likelihood maximization. In *UAI*.

Yu, C.; Chuang, J.; Computing, S.; Math, C.; Gerkey, B.; Gordon, G.; and Ng, A. 2005. Open-loop plans in multi-robot POMDPs. Technical report, Stanford CS Dept.