

Automated Model Approximation for Robotic Navigation with POMDPs

Devin Grady, Mark Moll, Lydia E. Kavraki

Abstract—Partially-Observable Markov Decision Processes (POMDPs) are a problem class with significant applicability to robotics when considering the uncertainty present in the real world, however, they quickly become intractable for large state and action spaces. A method to create a less complex but accurate action model approximation is proposed and evaluated using a state-of-the-art POMDP solver. We apply this general and powerful formulation to a robotic navigation task under state and sensing uncertainty. Results show that this method can provide a useful action model that yields a policy with similar overall expected reward compared to the true action model, often with significant computational savings. In some cases, our reduced complexity model can solve problems where the true model is too complex to find a policy that accomplishes the task. We conclude that this technique of building problem-dependent approximations can provide significant computational advantages and can help expand the complexity of problems that can be considered using current POMDP techniques.

I. INTRODUCTION

Motivation: Robot motion planning has advanced considerably, but accounting for uncertainty into planning is still very challenging [1]. A principled and general formulation that can handle this uncertainty is that of a Partially-Observable Markov Decision Process (POMDP). In the general case, POMDPs are computationally intractable (PSPACE-complete [2]), and are applied only to small problems [3]. However, ongoing research has successfully applied POMDPs to increasingly complex robotic tasks under uncertainty, such as grasping [4], navigation [5] and exploration [6], although the problem sizes remain fairly small.

Even state-of-the-art POMDP solvers (some recent examples being [4], [7]–[11]) may require hours of computation to find an optimal policy even for relatively small discrete problems with tens of states and less than 10 discrete actions that the robot can take. The policies they produce, however, are extremely useful in the robotics domain. Specifically, they encode an optimal solution, and can be computed off-line. The on-line computation then is a simple sequence of calls to a look-up table. Therefore, there is significant interest in improving the maximum problem size that can be considered with these methods.

Problem Definition: To reduce the computational burden of POMDPs, we introduce a method called Automated Model Approximation, or AMA. AMA builds a problem-specific approximation of the state and/or action spaces, to delay the curse of dimensionality, while attempting to maintain an approximation that can build a near-optimal policy.

To investigate the merit of this idea, we will focus on the robotic navigation problem under state and sensing uncertainty. The initial state of the robot is unknown, but is instead a belief spread uniformly across a set of states. Sensing uncertainty prevents the robot from ever collapsing this belief to a single state except in highly-constrained environments. The robot navigation task we will focus on is to take a sensor measurement while in any one of several goal states. Sensing is defined as its own action and therefore takes time, so an optimal policy should avoid sensing except when necessary. Finding the optimal balance between motion and sensing is very computationally expensive in the presence of uncertainty.

To reduce complexity of the discussion and implementation, we will assume that only the *action model*, defined as the state and action space product, will be modified by AMA. It is not, however, a restriction of the method, and state space modification will be investigated in the future. The action model is defined this way because it represents the selection of actions allowed in any particular region of the state space. Therefore, action models will comprise the primary input and the output of AMA.

AMA requires the definition of the simple action space, as well as a method to specify important states of where the true action complexity needs to be considered. The user will define the true action space, a simplified version of the action space, and an operator to augment the simple action model with the true complexity in specific regions of the state space. These three items implicitly define three action models: the simplified action space everywhere, the true action space everywhere, and the output of AMA, which is a mix of the prior two. For example, a simple action model may be moving with speed 1 in any direction, and the true action model also allows moving with speed 2. Then, AMA will build an action model where speed 2 is allowed in useful directions in regions of the state space that are probably going to be entered, but in all other areas only allows a speed of 1. By reducing the number of options globally available, it decreases the computational burden, and by using all options where needed, we allow the construction of near optimal policies for a given problem instance.

The output of AMA is a problem-specific estimation of the true action model. To evaluate the success of AMA, we give all three action models (simple, true, AMA) to a POMDP solver with the same task, and evaluate the utility of the action models based on the policies that the solver constructs.

Given a POMDP and initial belief as input, the POMDP solver will compute a globally optimal policy with respect to total reward that is achieved. This policy maps from observations to actions, maximizing the expected reward,

and should be valid over all possible observations. The POMDP solver will reason in the belief space, B , which has dimension equal to the size of the state space. A point $b \in B$ encodes the probability of being in each state. Thus for even a 10 by 10 discrete 2D grid of states, the space that must be reasoned over is of dimension 100. As opposed to much of the work in solving POMDPs, we do not set out to address the discrete nature of our robotic navigation problem definition, and work specifically with discrete state and action spaces. The discrete POMDP model is formally defined as (S, A, O, T, Ω, R) where

S = state space

A = action space

O = observation space

T = conditional transition probabilities

Ω = conditional observation probabilities

R = reward function: $(A, S) \rightarrow \mathbb{R}$

Many POMDP formulations include a discount factor, γ , that provides the rate at which future rewards are discounted. AMA does not require using a specific POMDP solver, although the computational benefits will be best if there is a way to explicitly disallow an action from a particular state so it will not be considered. This is not a requirement, however, because with an exceptionally high cost, most heuristics will avoid taking these actions.

Related Work: Because of the difficulty of solving a problem POMDP instance exactly, modern methods are generally point-based methods that approximate the solution using a small set of representative belief points [5], [12]–[14]. In this work, we utilize an existing point-based method, MCVI [7], to evaluate the action model produced using AMA. MCVI was selected because it is more direct to define different action models than the other solvers we looked at. AMA is not a new method to compute solution policies for POMDP models, but rather a method to construct a fast, medium fidelity model that is problem-specific and so can avoid high complexity in unnecessary regions.

AMA builds an approximate action model that is used to speed up POMDP computation. In doing so, it also builds an initial guess for a policy. This initial guess is used to bootstrap solving the AMA generated POMDP problem instance. Although this seems similar to applying a heuristic to the underlying POMDP solver, it only initializes the guess of the lower bound of expected reward better, and does not affect the sampling strategy of the POMDP solver. Many heuristics [8], [15]–[18] have been successfully applied to various POMDP problem instances, however we only utilize the extremely general sampling heuristic built in to MCVI. This avoids any interaction between a selected sampling heuristic and AMA’s action model computation. There is no reason to think that utilizing one of these sampling heuristics to improve solution speed would be problematic, but it would complicate the analysis of results. This is an open avenue for future research.

Hierarchical POMDP methods [9], [19], [20] are related to AMA in that they focus on building and solving multiple POMDP models that in turn solve a global problem definition. However, these methods are constructing decompositions of the action and state spaces. AMA, on the other hand, focuses on pruning and removing parts of the action cross state space to construct a useful approximation, rather than decomposing it. Thus, AMA could be incorporated with hierarchal methods.

Because the computational complexity of solving a POMDP problem instance grows exponentially with the size of the state space, some recent methods try to group similar states and actions together to maintain high simulation fidelity where needed [11], [21]. This work is similar in concept to our proposed method, but tends to require a strong set of assumptions and/or user-defined mapping functions to create these groups. In contrast, AMA requires definition of action models and an action model refinement function as opposed to mapping functions or the existence of stabilizing feedback controllers. Also, AMA does not attempt to provide an optimal approximation over the whole space, but instead only focuses on regions found to be important for a specific problem instance. The ideas of state and observation grouping are used to extend a POMDP solver to continuous spaces [11]. This modification to the underlying POMDP solver is generally orthogonal to the thrust of AMA, where we alter the cardinality of the spaces considered, depending on the results of solving a low accuracy model. Integrating these ideas could provide AMA extra information for model building, based on the groups constructed in the low accuracy solution.

Probably the closest idea to AMA is found in [22], which builds a decomposition of the state space with varying resolution depending on the environment features. However, AMA uses knowledge from an approximate solution to the specific problem instance, rather than only the environment specification. This means that AMA will retain a simple action model even where the environment is complex if those regions are not useful in the current problem instance. This is in contrast to a variable resolution method based on building quadrees over the entire space, where resolution depends on environment complexity only. Additionally, we focus on the action model, while [22] focused on the state space.

The work on policy transformation under changing models [23], called Point-Based Policy Transformation (PBPT) is extremely relevant although orthogonal to our approach. Specifically, AMA focuses on building an action model with the minimum complexity required, while PBPT focuses on making minimal modifications to a policy to fit a new model. AMA does not use such a sophisticated policy transformation, instead bootstrapping the new approximate solution with information about how the original, simple instance was solved. This simpler method requires fewer assumptions on the differences between the simple action model, the approximation of the true action model, and the true action model. In particular, PBPT requires a bijection between the (discrete) state, action and observation spaces between the two models under consideration, and our approach can not meet this requirement because we are explicitly changing the

cardinality of these spaces between the three action models.

Contributions: We introduce AMA as a general method to compute an approximate action model that can be used to find fast approximate solutions to POMDPs. Our high-level AMA framework is action, observation, and transition model agnostic, and our specific POMDP implementation for simulation results is presented in Section III. Given a robot model and environment that induce a belief space large enough to be computationally infeasible for existing POMDP solution methods, AMA probes the specific problem instance and automatically creates an action model that estimates where high fidelity is required to produce an optimal policy.

We investigate the performance benefit of AMA and discuss the relative merits of the action model approximation constructed by AMA as compared to the true action model. The use of the action model provided by AMA is shown to generally, although not always, improve the POMDP solution time while not dramatically impacting total expected reward. Simulations are performed over a range of environments to verify the scope of our results. Finally, we apply our method to an example system presented in the software library we used, and our initial results show the same trend of faster solution without quality decrease.

II. METHODOLOGY

AMA Framework: We will denote the true action model that defines the problem as M . Then let M^- be the simplified version of that model. The function $\hat{M} = \text{Refine}(M^-, R)$ is defined by the user and modifies the action model of M^- by adding new options in the product of state and action spaces, selected by analyzing the set of states R . That is, if an action model M^- and set of states R are passed into Refine, it shall return a new action model \hat{M} with additional complexity added. The analysis of R determines what actions are made available in what states. For example, if the simple model M^- only uses the 4 cardinal directions, and R includes turning a corner, then diagonal actions to short-cut that corner can be added. In general, if the states are nodes in a graph connected by action edges, then more complex models can be built with actions that, when added, will create shortcut paths in this graph. If all possible refinements are made to the simple action model M^- , the true action model M should result. Thus \hat{M} is an action model that has at least as much complexity as M^- , but no more complexity than M , and uses the information from the solution of the POMDP problem instance under M^- and the function Refine to increase action model complexity in only the relevant areas.

Algorithm 1 Automated model approximation algorithm

```

 $\hat{M} \leftarrow M^-$ 
for  $i = 1 \rightarrow \text{iterations}$  do
     $\text{policy} \leftarrow \text{Solve}(\hat{M})$ 
     $\text{states} \leftarrow \text{Execute}(\hat{M}, \text{policy})$ 
     $\hat{M} \leftarrow \text{Refine}(\hat{M}, \text{states})$ 
end for
return  $\hat{M}$ 

```

In Algorithm 1, Solve should call any POMDP solver, and return a policy that maximizes expected reward, as described in Section I. states is the set of ordered lists of true world states that the robot entered while executing policy . This is a set of lists because the simulation must account for all possible true initial conditions with the correct probability distribution. To accomplish this, 1000 samples are drawn from the initial belief and executed.

Our implementation of AMA then operates in an iterative fashion as described in Algorithm 1. We found this to be an intuitive way to pass information to Refine, although in general, any function of the current robot model and policy could work here. Then \hat{M} is updated with the information that this sequence of states provides. The exact form of the Refine function will differ between various POMDP models. Our implementation can be found in Section III.

III. IMPLEMENTATION

To implement AMA, we chose to work with a state-of-the-art POMDP solver known as MCVI [7]. The authors of MCVI provide an implementation of their method for download, and new problems are defined by writing C++ code. This enabled AMA implementation by separating out our AMA code into the action model implementation, and required less modification to the underlying code than may be the case with other POMDP solvers. MCVI is an iterative method, so it constructs a series of policies. The final policy will have the highest lower bound and lowest upper bound of expected reward. MCVI exits when upper bound – lower bound \leq convergence criterion. This criterion is 1 for all simulations.

In MCVI, the lower bound is found by particle filter simulation of the model applying a policy. Thus the actual reward of the policy may be lower than this lower bound, depending on the exact samples that get drawn in the particle filter. We will continue to use the term lower bound even though it has a dependence on randomness and may not be a strict lower bound for all samplings. (Similar arguments apply for the upper bound, although there is less variance here so it has a smaller effect.)

Problem Instance: In this section, we will define the spaces that the POMDP solver will be working in. The robot definition that we will concentrate on is a point robot living in a grid discretization of a 2D world. The robot state space thus is a set of square grid cells, and the action space is defined as noise free movement between cells. In this case, a clear choice for iterative refinement is over which diagonal edges are available. Our simple action model, M^- , can always move in the four cardinal directions, but not on diagonal edges. The true action model, M , is a model that can always move in the four cardinal directions as well as the four diagonal directions. Therefore, M^- should be able to solve all problems presented but the optimal path length could be larger by up to a factor of $\sqrt{2}$, as long as the environment does not include diagonal zero width corridors. We only present results in environments that fit this requirement. The Refine function can add diagonal actions where needed based on a set of execution traces (Algorithm 1), and is used in

AMA to create \hat{M} from M^- . The full complexity action model, M , is the baseline for comparison.

A POMDP model must provide costs or rewards for all actions possible. Moving in a cardinal direction and sensing always has cost 1, while moving diagonally has a cost $\sqrt{2}$. Sensing while in a goal region is the success condition for our task, so in this case the robot gets a large reward. MCVI uses a discount factor, and we leave it at the default value of $\gamma = .95$. The robot gets a null observation for every action except for the sense action. In this case, a two dimensional observation is provided that is sampled from a Gaussian around the current true state of the robot with variance three. The initial position of the robot is defined as a uniform sampling of a problem specified state and all adjacent non-obstacle states. This uncertainty in initial position drives the challenge of solving the problem, and the uncertainty in sensing prevents the system from collapsing belief to a single state. There is no penalty for attempting motion into an obstacle other than the lost time (discount of future reward) and the cost above.

Environments: AMA is fairly general in definition, but to evaluate the performance of the approximate robot models it constructs, we will define several concrete instances of problems to solve, as depicted in Figure 1.

- (a) A reasonable starting point is an empty environment, where the goal is simply to navigate from a point in the north to anywhere in the south. This is a useful example because it should not require diagonal actions to be optimal, it should run quickly, and the optimal policy should follow our intuition. Therefore we present it as a baseline for comparison. Solving \hat{M} should be fast and not take much time beyond the time to solve M^- , because the policy from M^- should have been near optimal already. So this environment should provide the best possible advantage to our method.
- (b,c) In the U environment, the robot starts inside a U shaped obstacle. In this case, some diagonal actions are needed for an optimal policy. We expect to see improvement with \hat{M} as compared to M , although not as much as the empty environment. We also test a smaller, more constrained version of this environment.
- (d) The Diagonal environment has a diagonal line cutting from the southwest up to just beyond the center. This is expected to be the hardest environment to solve because not only are there two very different navigation paths to follow, but the choice needs to be made soon without spending too much time sensing.
- (e) The Spikes environment has three narrow passages. This environment should solve quickly, because the narrow passages actually allow the robot to reduce the uncertainty in position. In addition, the starting location is fairly constrained, so the initial belief does not have as much spread as the other environments. The spikes are along the X and Y axes, so optimal navigation requires many diagonal actions.
- (f,g) The Maze environments require navigating four turns. The difference between the two mazes is that one is always as wide as the initial uncertainty, while the second

has one side as a narrow passage that should reduce the uncertainty part way through execution.

- (h) The final environment is for the underwater robot model in the MCVI package as found in [7]. In this model, the robot can only localize in the area on top and bottom (shaded lightly), while the goal region is to the right. A major difference in the model is that the true action space of M is smaller. Only five movement directions are available: north, northeast, east, southeast, and south. Therefore, the difference in action model complexity is significantly less than in our navigation model.

Iterative Refinement: In this section we will describe our implementation of $\text{Refine}(M^-, R)$ that is used in Algorithm 1 to construct \hat{M} . The first step is to find a solution policy with the simplest possible general action model, M^- . The simple action model, M^- , when passed to MCVI, should be able to solve any problem instance, although it need not be able to create an optimal policy with respect to the true model M . Because the policy generated from solving M^- is only used to initialize the next step, we define convergence as 10, rather than the (arbitrary, user defined) true convergence criterion of 1. Once this policy is computed, it is simulated 1000 times, and each execution trace is recorded in order to call the Refine function as described in Algorithm 1.

Because the POMDP solution under the action model M^- tends to yield policies that are optimal up to short-cutting corners, each state in the trace is checked to see if a sequence of diagonal actions can connect to a future state in the trace. If so, then \hat{M} will have that sequence of diagonal actions added to it. As an implementation detail, the center of the start distribution always has all 4 diagonal actions added to ensure that our initial belief is identical between \hat{M} and M , because it is initialized based on adjacency of regions.

Adding actions based on short-cuts tends to add exactly the edges that might be useful in the next iteration, but does not add extraneous edges that will increase computation time. In addition, the transitions are combined to create an initialization map and a new initial policy. This initial policy follows whatever transition was most likely from a given state in the previous iteration. In this way, the work done in the simpler action model can be applied in the more complex action model easily, while searching for improvements using the new diagonal actions available in \hat{M} . After processing all execution traces to create \hat{M} , it is then passed into the MCVI framework as a new problem instance, and it is run until convergence or timeout, with identical parameters to M . Our hypothesis is that a problem-specific action model \hat{M} will allow us to find a policy that is almost as good as the policy that the true action model provides, but in significantly less time. This iterative refinement could be applied multiple times, but in practice we found that AMA worked well with just one refinement step. Therefore there is no ambiguity in the use of M^- , \hat{M} , and M to describe the action models.

IV. RESULTS

Each environment was run in MCVI using our iterative refinement model on a single core to avoid timing discrep-

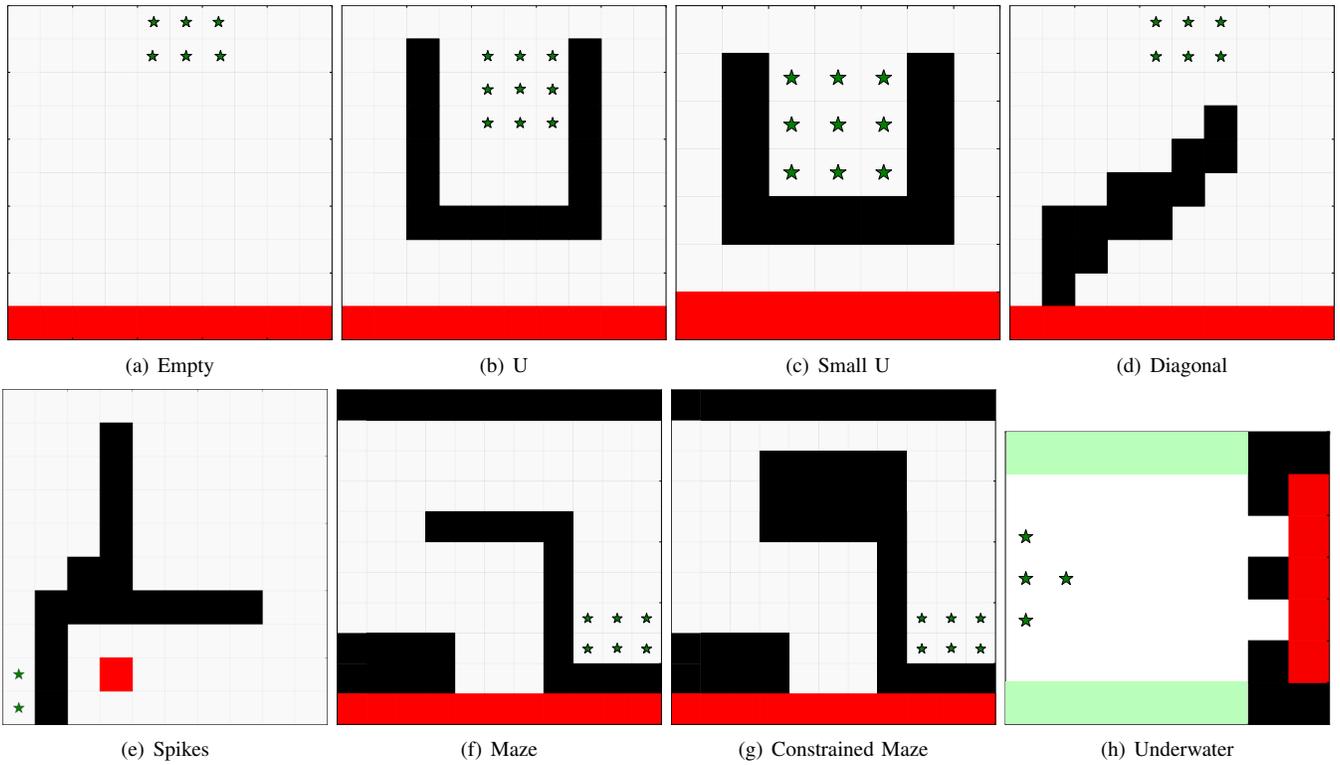


Fig. 1. Environments used for evaluation, where the black regions are hard obstacles, stars indicate an element of initial belief, and the light shaded area at the bottom is the goal region. The final environment is for the underwater robot model of [7], and has the goal region to the right while the shaded regions at top and bottom correspond to areas that allow localization.

ancies with varying levels of parallelism. All experiments were repeated 60 times to obtain statistically significant means to compare. All error bars represent 95% confidence intervals. The time limit for solving M^- and \hat{M} was 10,000 seconds, and the time limit for M was 20,000 seconds. Time taken to perform the refinement step is less than one second and is neglected here for clarity of plots. This time grows asymptotically linearly in the number of execution traces considered and the planning horizon, so it is not expected to ever be a significant influence on the total runtime.

Performance: The solver was able to converge to a correct solution in our test of the empty environment (Figure 1(a)), as our performance data shows in Figure 2. As expected, M^- followed by \hat{M} did very well, taking about 2500 seconds less than the time to solve M . As should be expected, increasing the size of the action space when the actions are not needed makes a large difference in computational power required. We see that the increase in time to solve both M^- and \hat{M} is smaller than the increase of time used due to additional actions available in M .

When the U shaped obstacle (Figure 1(b)) was tested, M and the hybrid model \hat{M} did not converge within the time limits, although the simpler M^- did. Therefore it is somewhat uninteresting to compare run-times, as they are simply pegged at timeout for everything except M^- . Thus we present a plot of the reward in Figure 3 of the best policy that the three cases were able to find. The results here are somewhat surprising – the solution using M did not return any policy that got a positive reward at all, while using the simpler M^- could,

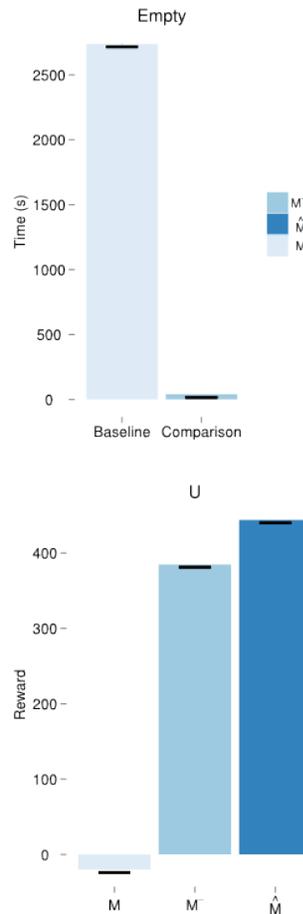


Fig. 2. In the empty environment, our approximate action model is more than an order of magnitude faster. The baseline is the true robot model M , while the comparison is the time to solve the model AMA built, \hat{M} , stacked on top of the time to solve the simple action model (M^-) that is used by AMA as a starting point. Note that the time for \hat{M} is so small as to be invisible on this plot.

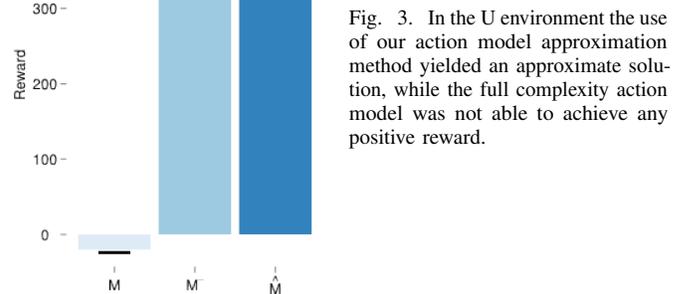


Fig. 3. In the U environment the use of our action model approximation method yielded an approximate solution, while the full complexity action model was not able to achieve any positive reward.

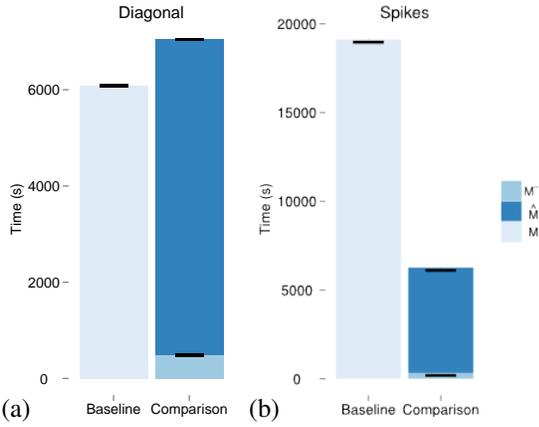


Fig. 4. The diagonal environment of Figure 1(d) caused our approximate models to use slightly more runtime, shown in (a). The spikes environment depicted in Figure 1(e) barely completed on time, and AMA provided a significant runtime advantage, shown in (b). Here, the time for solving M^- is hard to see because it is so small at the bottom of the comparison column.

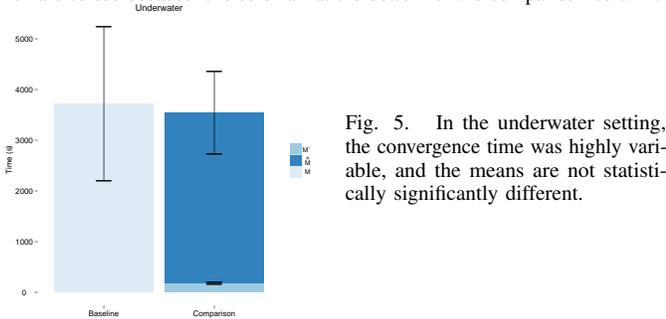


Fig. 5. In the underwater setting, the convergence time was highly variable, and the means are not statistically significantly different.

and then using \hat{M} was able to improve on the expected reward. Therefore, we see a significant benefit for AMA. It is apparent that solving with \hat{M} is able to provide partial answers quickly. In this case, using M failed to provide a reasonable solution at all. Because of the difficulty in solving this environment, we constructed a smaller version of it with about half the number of discrete states. However, the results were qualitatively identical: the full complexity action model M still did not yield a policy that got positive reward within 20,000 seconds in any of the 60 runs.

The environment with a diagonal line in it (see Figure 1(d)) was found to be able to converge in all cases. In this case, we see that the time of solving M actually beats the time to solve using M^- and \hat{M} combined, although not by a large margin, as shown in Figure 4(a). This fits our expectations, because M^- cannot provide as much information to the AMA, and there are many diagonal actions needed for an optimal policy. If we also look at the reward of the three methods in Figure 6(b), we can see again that using \hat{M} gets a value very close to when we use M , but clearly was missing out on some actions that were needed to be optimal. However, the absolute difference in reward is small.

The fourth environment with three spikes to navigate, depicted in Figure 1(e), also converged to a solution in all cases. The total time to solve, shown in Figure 4(b), is significantly less when using AMA. However, based on the number of diagonal actions required, it is reasonable to think that there will be degradation in the reward that \hat{M} was able

to achieve. The results presented in Figure 6(a) indicate that this is exactly what happens, a significant decrease in runtime is paid for by a smaller percentage decrease in reward.

The two maze environments of Figures 1(f),1(g) had varying results. As before, we will compare the reward that their partial execution could achieve, noting that in the constrained version of the problem, solving with the simple M^- was fast, but all other runs in these environments hit timeouts. We believe that the unconstrained maze is able to find a good solution with all three models (see Figure 5(c)) because the uncertainty in the start state doesn't matter as much. This is because there is enough room to make progress in an open-loop fashion without even trying to disambiguate between the possibilities for the true start state. The true state only significantly affects the policy in the vicinity of the goal. The true model is favored here because each corner taken must use diagonal actions to achieve an optimal policy, and the reward in the unconstrained case favors the true model that is not bootstrapped with a straight-line policy which can take extra time to show to be suboptimal. However, the constrained maze results (see Figure 6) are believed to be so different because the robot needs to effectively collapse the uncertainty in its Y position to reliably enter the narrow passage. The simpler models are able to do this within the time limits and therefore can get to the goal on average, while the true model is not able to find a policy that achieves this effective localization in the time limits provided.

Finally, when we tested AMA using a noisy underwater action model provided in MCVI as the true M , we found very high variance in runtime (see Figure 5), although the overall reward was found to be very consistent across the three models.

All trials completed and found a good policy, however, some of them took much longer than others to converge to that result. This was the case for both the true action model M and our intermediate complexity action model \hat{M} . The simple model, M^- , was much more consistently fast. Overall, the mean time using AMA was found to be less than the mean time to solve the true action model, although the variance present in the data prevents a definite result.

V. DISCUSSION

We presented a method for iterative refinement of action models in a POMDP setting, called Automated Model Approximation, for a class of robotic applications. AMA was applied to a discrete robot model and tested in several environments. In all but one environment, there was a significant computational benefit to using AMA. Either solution times were significantly reduced, or solutions were found in cases that the fully-refined model could not solve at all. AMA was also applied to an underwater navigation problem as presented by the authors of MCVI. Our method showed promise, but the results can not be considered statistically significant due to high variance in runtime. We believe that through refinement, solution times on many POMDPs, traditionally an extremely long computation, can be significantly reduced.

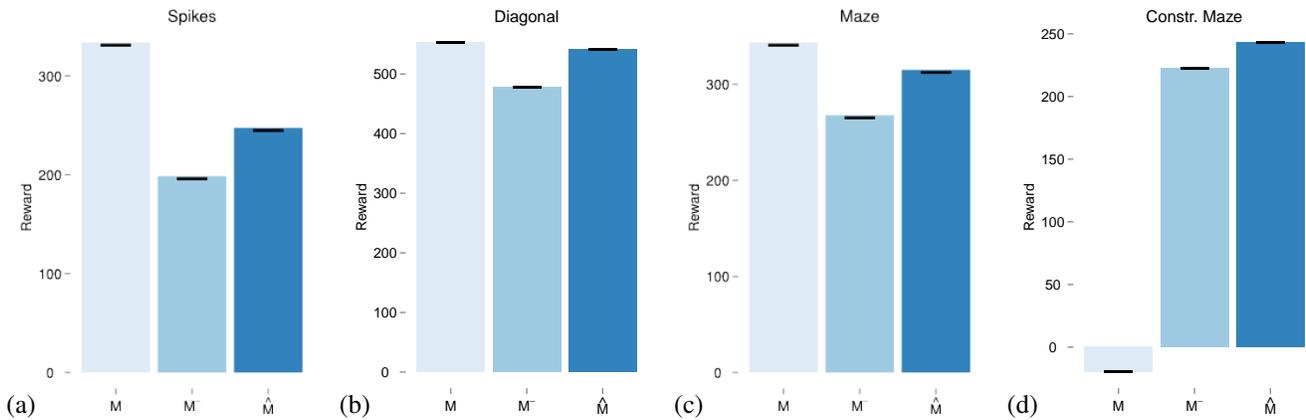


Fig. 6. In the spikes environment (a), the simpler models are able to run much faster, however, they miss some areas where additional complexity in the model was required to achieve optimality. The diagonal environment (b) produced a policy that achieved slightly lower reward. In the maze environments (c,d), the constrained version of the problem could not be solved by M , but the approximate models could; when unconstrained, both M and \hat{M} could be solved for approximately equal reward, noting that they all timeout so the total runtime for M and $\hat{M}+M^-$ are approximately equal at 20,000 seconds.

In the future, we will investigate the problems of variance associated with approximate methods such as MCVI, and try to automate the selection of good run-time parameters. Many other problems can be framed as a POMDP, and AMA may be useful in sensing problems such as target classification. State refinement is also interesting, and could be implemented in a grid-based state model by splitting grid cells into 4 smaller cells. Then the initial policy assumes all 4 sub-cells are equal – the solver will correct for cases where it is not. An extension of AMA in this manner could help increase the size of the state spaces that are considered in POMDP problems, in addition to the complexity of the action model.

ACKNOWLEDGEMENTS

This work was supported in part by the US Army Research Laboratory and the US Army Research Office under grant number W911NF-09-1-0383, NSF CCF 1018798, NSF IIS 0713623. D.G. is also supported by an NSF Graduate Research Fellowship. Equipment used was funded in part by the Data Analysis and Visualization Cyberinfrastructure funded by NSF under grant OCI-0959097.

REFERENCES

- [1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA: MIT Press, 2005.
- [2] C. Papadimitriou and J. Tsitsiklis, "The complexity of Markov decision processes," *Mathematics of Operations Research*, vol. 12, no. 3, pp. 441–450, 1987.
- [3] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1-2, pp. 99–134, May 1998.
- [4] K. Hsiao, L. P. Kaelbling, and T. Lozano-Perez, "Grasping POMDPs," in *Proc. 2007 IEEE Intl. Conf. on Robotics and Automation*. Ieee, Apr. 2007, pp. 4685–4692.
- [5] N. Roy and S. Thrun, "Coastal navigation with mobile robots," in *Advances in Neural Processing Systems*, 1999, pp. 1043–1049.
- [6] T. Smith and R. Simmons, "Point-based pomdp algorithms: Improved analysis and implementation," in *UAI*. AUAI, 2005, pp. 542–547.
- [7] Z. W. Lim, D. Hsu, and L. Sun, "Monte carlo value iteration with macro-actions," in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, Eds., 2011, pp. 1287–1295.
- [8] J. S. Dibangoye, A.-I. Mouaddib, and B. Chai-draa, "Point-based incremental pruning heuristic for solving finite-horizon DEC-POMDPs," in *Intl. Conf. on Autonomous Agents and Multiagent Systems*. Budapest, Hungary: Intl. Foundation for Autonomous Agents and Multiagent Systems, 2007, pp. 569–576.
- [9] A. Foka and P. Trahanias, "Real-time hierarchical POMDPs for autonomous robot navigation," *Robotics and Autonomous Systems*, vol. 55, no. 7, pp. 561–571, July 2007.
- [10] H. Kurniawati, D. Hsu, and W. S. Lee, "SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces," in *Robotics: Science and Systems*, 2008.
- [11] H. Kurniawati, T. Bandyopadhyay, and N. Patrikalakis, "Global motion planning under uncertain motion, sensing, and environment map," *Autonomous Robots*, vol. 33, pp. 255–272, 2012.
- [12] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for POMDPs," *Intl. Joint Conf. on Artificial Intelligence*, vol. 18, pp. 1025–32, 2003.
- [13] H. Kurniawati, Y. Du, D. Hsu, and W. S. Lee, "Motion planning under uncertainty for robotic tasks with long time horizons," *The Intl. Journal of Robotics Research*, 2010.
- [14] H. Bai, D. Hsu, W. Lee, and V. Ngo, "Monte carlo value iteration for continuous-state pomdps," in *Algorithmic Foundations of Robotics IX*, ser. Springer Tracts in Advanced Robotics, D. Hsu, V. Isler, J.-C. Latombe, and M. Lin, Eds. Springer Berlin / Heidelberg, 2011, vol. 68, pp. 175–191.
- [15] Z. Zhang and X. Chen, "Accelerating Point-Based POMDP Algorithms via Greedy Strategies," *Simulation, Modeling, and Programming for Autonomous Robots*, vol. 6472, pp. 545–556, 2010.
- [16] M. Hauskrecht, "Value-Function Approximations for Partially Observable Markov Decision Processes," *Journal of Artificial Intelligence Research*, vol. 13, pp. 33–94, June 2000.
- [17] O. Madani, S. Hanks, and A. Condon, "On the undecidability of probabilistic planning and infinite-horizon partially observable markov decision problems," in *Proc. of the sixteenth national conference on Artificial intelligence*, ser. AAAI '99. Menlo Park, CA, USA: American Association for Artificial Intelligence, 1999, pp. 541–548.
- [18] W. S. Lovejoy, "A survey of algorithmic methods for partially observed Markov decision processes," *Annals of Operations Research*, vol. 28, no. 1, pp. 47–65, Dec. 1991.
- [19] J. Pineau, N. Roy, and S. Thrun, "A hierarchical approach to POMDP planning and execution," *Workshop on Hierarchy and Memory in Reinforcement Learning*, 2001.
- [20] J. Pineau, M. Montemerlo, M. Pollack, N. Roy, and S. Thrun, "Towards robotic assistants in nursing homes: Challenges and results," *Robotics and Autonomous Systems*, vol. 42, no. 3-4, pp. 271–281, Mar. 2003.
- [21] A.-a. Agha-mohammadi, S. Chakravorty, and N. Amato, "FIRM: Feedback controller-based information-state roadmap - A framework for motion planning under uncertainty," in *Intl. Conf. on Intelligent Robots and Systems*. IEEE, Sept. 2011, pp. 4284–4291.
- [22] R. Kaplow, A. Atrash, and J. Pineau, "Variable resolution decomposition for robotic navigation under a POMDP framework," in *IEEE Intl. Conf. on Robotics and Automation*. IEEE, May 2010, pp. 369–376.
- [23] H. Kurniawati and N. M. Patrikalakis, "Point-Based Policy Transformation : Adapting Policy to Changing POMDP Models," in *Algorithmic Foundations of Robotics X*, ser. Springer Tracts in Advanced Robotics, D. Hsu, V. Isler, J.-C. Latombe, and M. Lin, Eds. Springer Berlin / Heidelberg, 2012, pp. 1–16.