Improved POMDP Tree Search Planning with Prioritized Action Branching

John Mern,[1] **Anil Yildiz,** [1] **Larry Bush** [2] **Tapan Mukerji** [3] **Mykel J. Kochenderfer** [1]

[1]Stanford University, Department of Aeronautics and Astronautics, 496 Lomita Mall, Stanford, CA 94305
[2]General Motors, Research and Development, Warren, MI
[3]Stanford University, Department of Energy Resources Engineering, 367 Panama Street, Stanford, CA 94305
{jmern91, yildiz, mukerji, mykel}@stanford.edu
bushL2@alum.mit.edu

Abstract

Online solvers for partially observable Markov decision processes have difficulty scaling to problems with large action spaces. This paper proposes a method called PA-POMCPOW to sample a subset of the action space that provides varying mixtures of exploitation and exploration for inclusion in a search tree. The proposed method first evaluates the action space according to a score function that is a linear combination of expected reward and expected information gain. The actions with the highest score are then added to the search tree during tree expansion. Experiments show that PA-POMCPOW is able to outperform existing state-of-the-art solvers on problems with large discrete action spaces.

Introduction

Sequential decision making problems under uncertainty are often modeled as partially observable Markov decision processes (POMDPs) (Littman, Cassandra, and Kaelbling 1995). A solution to a POMDP is a policy that maps a belief over the state of the environment to an optimal action that maximizes the sum of discounted rewards over a series of steps. Solving POMDPs exactly is generally intractable and has been shown to be *PSPACE-complete* for finite horizons (Papadimitriou and Tsitsiklis 1987). Therefore, a variety of offline and online approximate solution methods have been proposed (Kochenderfer 2015).

Offline solvers compute the full policy before any action is taken, and are typically effective at small to moderately sized POMDPs (Ross et al. 2008). Monte-Carlo methods using point-based belief space interpolation were initially explored (Thrun 1999). Many advanced solvers now use point-based value iteration to learn an approximation to the belief value function from a finite-set of belief points (Kurniawati, Hsu, and Lee 2008). However, because offline solvers compute policies over the entire belief space, they are typically not viable for large problems.

Several online planners have been developed by adapting Monte-Carlo Tree Search (MCTS) for partially observable environments. Because online planners only reason about beliefs reachable from the current belief, they can typically be applied to much larger problems (Silver and Veness 2010; Somani et al. 2013; Sunberg and Kochenderfer 2018).

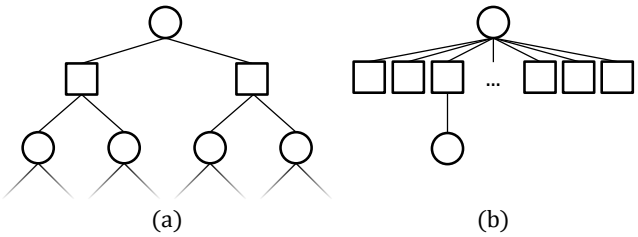


Figure 1: MCTS trees. (a) A deep search tree with a small action space. Action nodes (squares) are sampled frequently. (b) A shallow search tree with a large action space.

The POMCP algorithm (Silver and Veness 2010) adapted UCT search (Kocsis and Szepesvári 2006) by using generative models and sampling states from an unweighted particle set to search over action-observation trajectories. The DESPOT algorithm (Somani et al. 2013) takes a similar approach, using a deterministic generative model to reduce the tree search variance. The ABT algorithm (Kurniawati and Yadav 2013) was proposed to improve planning speed by reusing part of the previous belief step search tree.

Existing online methods may still fail when the action space of the problem is very large, such as in large-scale route planning or robotic control. During tree search, the probability of sampling a given action from a large space is very low, resulting in wide, shallow search trees (Sunberg and Kochenderfer 2018), as shown in Figure 1. In MCTS methods, shallow trees provide poor estimates of the action values (Silver and Veness 2010).

To scale to problems with larger action and observation spaces, the POMCPOW and PFT-DPW algorithms (Sunberg and Kochenderfer 2018) introduce double progressive widening (DPW) to POMCP. Progressive Widening (Couëtoux et al. 2011) was introduced to scale MCTS methods to large discrete and continuous spaces by dynamically limiting the number of action nodes added during tree expansion. *Double* progressive widening applies the progressive widening to both the action and observation spaces. DPW has been shown to be sensitive to the order nodes are selected for addition (Browne et al. 2012) and has limited effect on scaling to large or multidimensional action spaces.

We propose a method to select a subset of the most

ISSN: 2692-5079

fuel in order to contain the spread of the wildfire. Fire propagation is probabilistic, according to dynamics defined in a previous work (Julian and Kochenderfer 2019). This task was designed to test the performance of PA-POMCPOW on a task with non-stationary dynamics and a sparse reward.

In this model, fire starts at some points in the grid and burns at those points until the fuel is exhausted. A fuel-containing cell that is currently not burning ignites with probability proportional to the number of its neighbors currently burning. The fire is able to spread beyond immediately adjacent cells, up to two cells away. Wind biases the fire propagation direction and changes randomly each step.

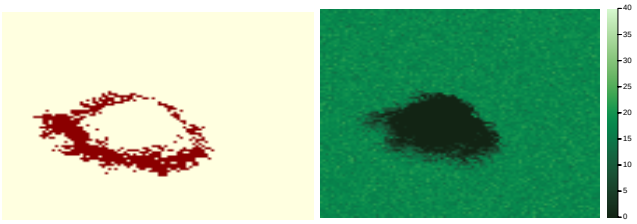


Figure 2: Wildfire containment task. The left figure shows the burn map, where dark areas are currently burning and light are un-ignited. The right figure shows the fuel map, where lighter values correspond to higher fuel levels. A southwest wind biases fire propagation.

The state is represented by a burn map, a fuel map, and a wind vector, as shown in Figure 2. The burn map is an array of which cells in the grid world are on fire. The fuel map is an array of how much fuel is contained in each cell and is generated by sampling each cell from a truncated Gaussian distribution. Wind is uniform over the grid and the vector is sampled from a 2D uniform distribution between $[-1, 1]$.

Areas in each corner of the grid are designated to be keep-out areas. Associated with each area is a counter c_i which decreases at each time step until it reaches zero. The objective of the task is to keep the fire from reaching any keep out zone until the zone counter is zero. If fire reaches a keep-out zone, that zone's counter is set directly to zero and a reward equal to the remaining counter value, $-c_i$. The episode terminates when all zones have a zero count.

At each step, the agent selects a non-burning grid cell to clear of fuel. The fuel level in the selected cell and eight surrounding cells are then set to zero. The wind vector is updated by addition of zero-mean Gaussian noise.

The agent has full knowledge of the burn map, fuel map, and keep-out counters. The agent also makes noisy measurements of the wind. The noise on the wind measurement is proportional to the distance between the action location and the fire. Fire reaching any keep-out zone has a cost equal to ten times the counter value of the keep-out zone.

To solve this task, we used a Gaussian distribution to model the wind belief, and updated it using a Kalman filter (Kalman 1960). The Λ set was composed of linearly spaced values between 0.5 and 1.5 with a step size of 0.1 for a total of 16 values. The linear-Gaussian forms of the score function were used, however, because the reward was sparse, a dense, shaped reward was implemented. The shaped reward function was defined as $r(a, s) = \theta d_f(a) + \beta d_k(a)$, where d_f measures the distance of the cleared cell to the closest burning cell and $d_k(a)$ is the distance to the nearest keep-out zone. The θ and β terms are weighting factors.

We ran each test with limits of 100, 250, and 500 simulator calls per solver step for a grid size of 40×40. For each run, we recorded the total accumulated reward and the average planner run time per step. As with the sensor placement task, we ran each test for the POMCP and POMCPOW baseline algorithms as well, using the same belief distribution and number of solver calls.

We additionally implemented a myopic expert policy baseline that clears all the fuel bordering each keep-out zone. At each step, the policy clears the cells immediately bordering a keep-out zones in order to create a barrier around the zone. The policy chooses to clear cells of the keep-out zone that is closest to the fire, until all the bordering cells have been cleared. It then moves to the next closest zone.

Results

Sensor Placement

The performance of each algorithm on the sensor placement task is reported in Table 1. The mean score and standard error over the 100 trials are reported for each test point.

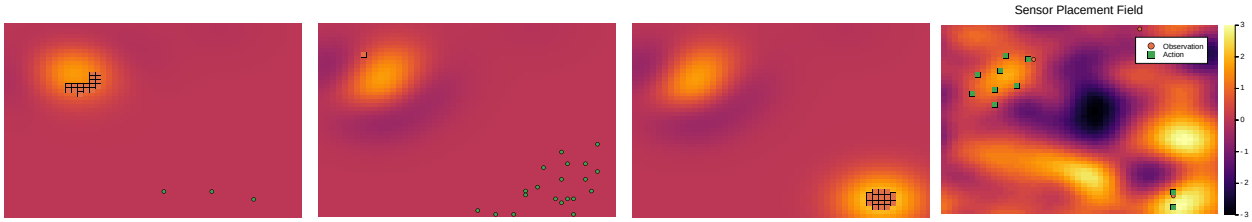
PA-POMCPOW outperformed the baseline algorithms for all test points. It also outperformed the Greedy policy in all but one test point, showing that with action subsets, the tree search is still able to find non-myopic policies. Neither baseline outperformed the greedy policy in any case.

The relative gap between POMCPOW and Greedy remained at approximately 30% for 1000 queries at all grid sizes. This seems to suggest that the shallow trees generated by POMCPOW resulted in selection of the approximately greedy action. This also suggests why PA-POMCPOW, with deeper search trees, was able to outperform it.

On average, PA-POMCPOW required $4.2\,\mathrm{ms}$, $8.3\,\mathrm{ms}$, and $43.5\,\mathrm{ms}$ per simulation call for the 20×20 grid, 50×50 grid, and 100×100 grid respectively. This was significantly more expensive than the most efficient baseline, POMCPOW, which required $0.7\,\mathrm{ms}$, $2.4\,\mathrm{ms}$, $10.1\,\mathrm{ms}$ for the three respective grids. This is likely due to the added cost of computing the action scores at each observation node.

A partial episode is shown in Figure 3. The location and type of each root node action of the search tree is shown overlaid on the Gaussian process belief mean values. A balance of exploration and exploitation is seen over the episode, however, when high expected reward actions are available, the actions tend to form tight clusters, which may not be desirable in some tasks.

From the tree measurement experiment, the average maximum node depth and standard error was 8.13 ± 0.23, 3.61 ± 0.06, and 3.05 ± 0.05 for PA-POMCPOW, POMCPOW, and POMCP, respectively. The average maximum depth was significantly higher for PA-POMCPOW than either baseline. These results suggest that improved tree depth contributed to PA-POMCPOW's improved performance on the task.



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

Figure 3: Example sensor placement action branching. The first three figures show the Gaussian process mean values with root node actions overlaid. The squares mark sensor placement actions and the circles mark observation only actions. (a) The algorithm prefers actions with high known reward early in the episode. (b) As the rewarding actions are depleted, the algorithm considers more exploration. (c) The algorithm once again prefers high-reward actions once a new high-value area is found. (d) The actual information state for the episode and selected actions at episode completion.

Table 1: Sensor Placement Task Scores

Grid Size	Calls	**PA-POMCPOW**	POMCPOW	POMCP	Greedy
20×20	100	3.35 ± 0.22	0.77 ± 0.22	1.31 ± 0.24	2.55 ± 0.21
	500	3.66 ± 0.21	1.45 ± 0.23	1.26 ± 0.20	
	1000	$\mathbf{4.04 \pm 0.21}$	1.70 ± 0.26	1.18 ± 0.23	
50×50	100	$\mathbf{5.79 \pm 0.29}$	0.42 ± 0.24	1.40 ± 0.25	5.10 ± 0.32
	500	5.64 ± 0.28	2.90 ± 0.28	0.96 ± 0.23	
	1000	5.46 ± 0.29	3.80 ± 0.38	0.48 ± 0.26	
100×100	100	6.45 ± 0.44	3.99 ± 0.41	3.36 ± 0.44	6.69 ± 0.40
	500	7.68 ± 0.41	5.64 ± 0.43	3.10 ± 0.41	
	1000	$\mathbf{7.77 \pm 0.44}$	5.57 ± 0.43	3.10 ± 0.39	

Table 2: Wildfire Task Loss

Calls	Algorithm	Loss
100	PA-POMCPOW	$\mathbf{460 \pm 46}$
	POMCPOW	937 ± 24
	POMCP	1021 ± 30
250	PA-POMCPOW	$\mathbf{434 \pm 46}$
	POMCPOW	897 ± 33
	POMCP	1000 ± 28
500	PA-POMCPOW	$\mathbf{430 \pm 43}$
	POMCPOW	798 ± 29
	POMCP	1011 ± 28
–	Expert	722 ± 18

Wildfire Containment

The performance results from the wildfire containment task are shown in Table 2. As with the sensor placement task, PA-POMCPOW was able to outperform both of the baseline algorithms and the expert policy in all three test scenarios. The expert policy outperformed both baselines.

The computational cost of the wildfire task was slightly higher than that of the sensor placement task. The time per-query was $11.6\,\text{ms}$ for PA-POMCPOW, $8.5\,\text{ms}$ for POMCPOW, and $8.2\,\text{ms}$ for POMCP. As before, PA-POMCPOW was more expensive than POMCPOW and POMCP.

Despite the more complex environment and sparse reward function, PA-POMCPOW was still able to solve the problem better than the existing state-of-the-art and an expert policy.

Conclusions

We presented a general method to extend online POMDP solvers to problems with very large action spaces by prioritizing actions for tree expansion. Specific formulations of the method for various reward functions and belief distributions were presented. We implemented this method as a new algorithm called Prioritized Action POMCPOW (PA-POMCPOW) which can scale to very large problems.

The current work is limited to problems in which the score function terms can be analytically formed or easily and accurately approximated. Future work will investigate more generally extensible functions for the exploration and exploitation terms of the action score.

This work presented a static method of selecting the action subset. That is, once selected, the action space subset was never updated. Future work will explore dynamically adjusting the subset based on the action-value estimates.

This work also only directly considered large, discrete action spaces. While the proposed methods can be applied to continuous spaces in principle, evaluating the action score in a continuous domain would likely be intractable for many problems. Because of this, future work will investigate using the volume coverage metrics for action clustering (Kurniawati, Hsu, and Lee 2008).

Despite these limitations, experimental results showed that PA-POMCPOW was effective on very large problems. Using the proposed method with DPW improved the performance over existing state-of-the-art methods.

References

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time Analysis of the Multiarmed Bandit Problem. *Journal of Machine Learning Research* 47(2-3): 235–256. doi: 10.1023/A:1013689704352.

Browne, C.; Powley, E. J.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Liebana, D. P.; Samothrakis, S.; and Colton, S. 2012. A Survey of Monte

Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games* 4(1): 1–43. doi:10.1109/TCIAIG.2012.2186810.

Couëtoux, A.; Hoock, J.; Sokolovska, N.; Teytaud, O.; and Bonnard, N. 2011. Continuous Upper Confidence Trees. In *Learning and Intelligent Optimization (LION)*. doi:10.1007/978-3-642-25566-3_32.

Cover, T. M.; and Thomas, J. A. 2006. *Elements of information theory (2. ed.)*. Wiley. ISBN 978-0-471-24195-9. URL http://www.elementsofinformationtheory.com/.

Egorov, M.; Sunberg, Z. N.; Balaban, E.; Wheeler, T. A.; Gupta, J. K.; and Kochenderfer, M. J. 2017. POMDPs.jl: A Framework for Sequential Decision Making under Uncertainty. *Journal of Machine Learning Research* 18: 26:1–26:5.

Frazier, P. I. 2018. A Tutorial on Bayesian Optimization. *Computing Research Repository* .

Julian, K. D.; and Kochenderfer, M. J. 2019. Distributed Wildfire Surveillance with Autonomous Aircraft Using Deep Reinforcement Learning. *AIAA Journal of Guidance, Control, and Dynamics* 42(8): 1768–1778. doi:10.2514/1.G004106.

Kalman, R. E. 1960. A New Approach to Linear Filtering and Prediction Problems. *ASME Journal of Basic Engineering* 82: 35–45.

Kochenderfer, M. 2015. *Decision Making Under Uncertainty: Theory and Application*. MIT Press. ISBN 9780262029254.

Kochenderfer, M.; and Wheeler, T. 2019. *Algorithms for Optimization*. MIT Press. ISBN 9780262039420.

Kocsis, L.; and Szepesvári, C. 2006. Bandit Based Monte-Carlo Planning. In *European Conference on Machine Learning (ECML)*. doi:10.1007/11871842_29.

Kurniawati, H.; Hsu, D.; and Lee, W. S. 2008. SARSOP: Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces. In *Robotics: Science and Systems IV*. doi:10.15607/RSS.2008.IV.009.

Kurniawati, H.; and Yadav, V. 2013. An Online POMDP Solver for Uncertainty Planning in Dynamic Environment. In *International Symposium on Robotics Research*. doi:10.1007/978-3-319-28872-7_35.

Littman, M. L.; Cassandra, A. R.; and Kaelbling, L. P. 1995. Learning Policies for Partially Observable Environments: Scaling Up. In *International Conference on Machine Learning (ICML)*. doi:10.1016/b978-1-55860-377-6.50052-9.

Papadimitriou, C. H.; and Tsitsiklis, J. N. 1987. The complexity of Markov decision processes. *Mathematics of Operations Research* 12(3): 441–450.

Petersen, K. B.; and Pedersen, M. S. 2008. *The Matrix Cookbook*. Technical University of Denmark. URL http://www2.imm.dtu.dk/pubdb/p.php?3274.

Rasmussen, C. E.; and Williams, C. K. I. 2006. *Gaussian Processes for Machine Learning*. MIT Press.

Ross, S.; Pineau, J.; Paquet, S.; and Chaib-draa, B. 2008. Online Planning Algorithms for POMDPs. *Journal of Artificial Intelligence Research* 32: 663–704. doi:10.1613/jair.2567.

Silver, D.; and Veness, J. 2010. Monte-Carlo Planning in Large POMDPs. In *Advances in Neural Information Processing Systems (NIPS)*.

Somani, A.; Ye, N.; Hsu, D.; and Lee, W. S. 2013. DESPOT: Online POMDP Planning with Regularization. In *Advances in Neural Information Processing Systems (NIPS)*.

Sunberg, Z. N.; and Kochenderfer, M. J. 2018. Online Algorithms for POMDPs with Continuous State, Action, and Observation Spaces. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Thrun, S. 1999. Monte Carlo POMDPs. In *Advances in Neural Information Processing Systems (NIPS)*.