Hierarchical Monte Carlo Tree Search for Tethered AUV Planning

Gabriel B Margolis [1]

Abstract

Monte Carlo Tree Search (MCTS) is an asymmetric tree search algorithm which uses value estimates, obtained from random sampling, to focus its search in high-reward areas of state space. Multi-agent collaborative planning, in which multiple agents must simulataneously select actions to maximize a joint reward function, is a promising application area of MCTS. However, planning over long time horizons with MCTS is a challenge, because the number of action sequences to be considered grows exponentially in the time horizon. Hierarchical modeling can enable long-horizon planning with shallower search trees, by the introduction of temporally abstracted macro-actions. We apply Hierarchical Monte Carlo Tree Search to a multi-agent scenario where a monthership coordinates with a tethered AUV to collect reward.

1. Motivation

We consider the problem of generating complete plans for a multi-agent system containing a mothership and tethered AUV in an underwater volcanic environment. In this system, an effective plan must take into consideration the short-scale motions of the AUV as well as the large-scale trajectory of the mothership. If the AUV's motions are not planned in fine detail, it will fail to collect reward even when the mothership steers it into reward-rich regions. If the mothership's motions over a long timescale are not well-planned, the AUV will be forced into regions that contain little reward to collect.

Modeling and solving the tethered AUV problem using traditional or 'flat' MCTS is intractable for all but the smallest domains. Although MCTS provides significant speedup over naive search algorithms like breadth-first search, it still suffers from exponential slowdown with increased planning horizon length. Additionally, the sparser and distant the reward is in a given domain, the weaker the reward signal obtained is, and the less asymmetry the MCTS search tree achieves, worsening performance.

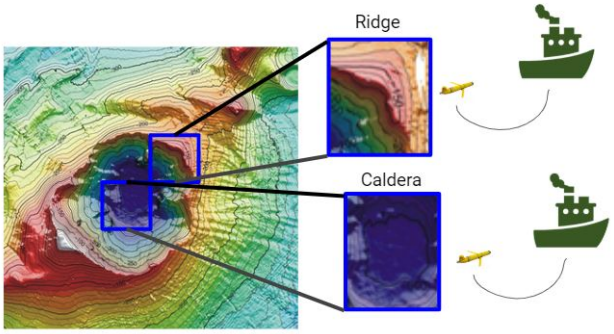

Figure 1. Tethered AUV planning in an underwater volcanic environment. The mothership (green) enables fast traversal of the environment; the tethered AUV (yellow) performs slow but precise reward-gathering actions.

2. Problem Statement

Our problem setting consists of a single mothership and single tethered AUV, which cooperate to explore an underwater volcanic environment (Figure 1).

Our AUV moves through a finely discretized grid-world using four translational actions, each representing motion for a single unit of time in a cardinal direction. Reward is obtained when the AUV first travels through a grid cell. All the reward in a grid cell is consumed once collected by the AUV and cannot be collected multiple times.

Our mothership moves through a coarsely discretized grid-world using four translational actions, each representing motion for two units of time in a cardinal direction, and a fifth action which represents the deployment of an AUV for ten units of time in the current location. Each large cell of our mothership's grid-world corresponds to a 10-by-10 grid of small cells in the AUV's grid-world discretization, and the AUV is constrained to remain within the same large cell as the mothership while it is deployed. Our mothership moves five times faster than its tethered AUV and is assumed

[1]Massachusetts Institute of Technogy, Cambridge, Massachusetts, USA. Correspondence to: Gabriel B Margolis <gmargo@mit.edu>.

to remain still while the AUV is deployed. The mothership does not directly collect reward.

Our mothership and AUV jointly plan to maximize the reward collected by the AUV over a fixed time horizon of 70 units of time.

To further constrain our example, we assume that each cell of our mothership's grid-world contains one of two types of region: a caldera or a ridge. We assume that regions of the same type contain the same reward distribution for the AUV, so that a given AUV behavior produces the same reward in the same type of region.

3. Background

Monte Carlo Tree Search is an asymmetric, anytime search algorithm which uses the rewards obtained from simulated rollout samples as a heuristic to guide the construction of its search tree. Applied to planning, each node of an MCTS search tree represents a state, and keeps track of that state's average rollout reward and visitation count. The edges of the MCTS search tree represent actions which transition between the states they connect. A single stage of MCTS consists of four steps:

Selection. Actions within the existing search tree are selected using a tree policy balancing exploration and exploitation, such as the Upper Confidence Bound.

Expansion. Once a state is encountered which is not already represented on the search tree, it is initialized as a node and added to the tree.

Simulation. A rollout policy (e.g. random action selection) is applied, starting at the newly expanded state and terminating when a stopping condition (e.g. fixed time horizon) is reached. States achieved during this step are not added to the search tree.

Backpropagation. The cumulative reward obtained in the terminal state is used to update the value estimates of all ancestor nodes in the search tree, along with their visitation counts.

Monte Carlo Tree Search has been applied to plan in multi-agent environments. The winner of the Ms. Pac-Man vs Ghost Team Competition, a multi-agent planning challenge presented at the IEEE Congress on Evolutionary Computation in 2011, was one of the earliest applications of MCTS to a multi-agent environment (Nguyen & Thawonmas, 2013). More recently, extensions of MCTS to multi-agent planning settings with additional constraints, such as limited communication, have been an active area of research (Best et al., 2018).

Hierarchical Monte Carlo Tree Search has been previously demonstrated to improve planning performance in various settings. Vien & Toussaint (2015) introduced the H-UCT algorithm, which assembles subtrees for primitive action selection together with a meta-tree which plans over macro-actions. The authors demonstrated H-UCT's improved performance in a classic single-agent hierarchical domain, the Taxi environment. Kurzer et al. (2018) extended hierarchical MCTS to a multi-agent collaborative setting, defining macro-actions for autonomous vehicles to perform coordinated maneuvers.

4. Method

We formulate the tethered AUV problem as a hierarchical planning problem. We define two types of macro-actions which arise naturally from the structure of this environment. The first type of macro-action is the deployment of the AUV from the mothership followed by a sequence of primitive AUV actions. The second type of macro-action is the motion of the mothership (this is also a primitive action).

4.1. Flat MCTS

Our baseline algorithm for solving the tethered AUV planning problem does not exploit any hierarchical structure. We simple build a search tree over primitive actions using the flat MCTS algorithm. This is referred to as Flat MCTS.

4.2. Naive Hierarchical MCTS

Our naive hierarchical MCTS algorithm (Algorithm 1) exploits the strict hierarchical structure of the tethered AUV planning problem aggressively. First, Naive Hierarchical MCTS learns a sequence of primitive actions which best achieves each defined macro-action. Then, it performs MCTS over macro-actions to produce a complete policy.

Although Naive Hierarchical MCTS is useful in our tethered AUV environment, its utility is limited in more complex environments. This is because at initialization it constructs a full search tree for every possible macro-action. In our environment, there are only sixteen such actions (the size of our grid). Depending how macro-actions are defined, a larger problem might have many more. Naive Hierarchical MCTS essentially amounts to an exhaustive search over macro-actions, and the entire reason we plan using MCTS is to avoid exhaustive search.

4.3. Online Hierarchical MCTS

Online Hierarchical MCTS avoids an exhaustive evaluation of macro-actions by simultaneously learning which macro-actions to execute and how to execute them. While naive hierarchical MCTS fully evaluates all macro-actions at initialization, Online Hierarchical MCTS partially evaluates a macro-action each time it is taken in the meta-tree. Al-

Algorithm 1 Naive Hierarchical MCTS

Initialize rewards for the entire map, following different probability distributions of reward in the Caldera and Ridge regions.
for x_M in macro_actions **do**
  x_{AUV} = x_M.Start
  sub_tree = MCTS(x_{AUV}, primitives, rewards)
  sub_tree.search(rollouts=1000)
  mrewards[x_M] = sub_tree.maxReward
end for
meta_tree = MCTS($x_{M,0}$, macro_actions, mrewards)
meta_tree.search(rollouts=1000)

Algorithm 2 Online Hierarchical MCTS

function onlineExecuteMacroAction(m)
  sub_trees[m].search(rollouts=10)
  executeMacroAction
end function
Initialize rewards for the entire map, following different probability distributions of reward in the Caldera and Ridge regions.
for x_M in macro_actions **do**
  x_{AUV} = x_M.Start
  sub_trees[x_M] = MCTS(x_{AUV}, primitives, rewards)
end for
meta_tree = MCTS($x_{M,0}$, macro_actions, mrewards)
meta_tree.search(rollouts=1000)

though this has the drawback of introducing nonstationary reward into the meta-tree, it confers the benefit that subtrees are grown proportional to the amount that they are selected in the meta-tree - so, if a macro-action is not taken in the meta-tree during planning, it is never evaluated, and frequently-taken macro-actions are evaluated for more rollouts than rarely-taken ones.

5. Implementation Details

Our hierarchical MCTS implementation extended the framework built by our team for our Advanced Lecture problem set and Grand Challenge implementation. We implemented new state and action objects for the mothership agent, with a different action set and state representation than our glider. We also modified our core MCTS algorithm to enable the use of the online hierarchical MCTS rollout policy, which takes advantage of the meta-policy search tree saved from past executions of the current macro-action.

6. Evaluation

We evaluate our three methods: Flat MCTS, Naive Hierarchical MCTS, and Online Hierarchical MCTS, in our simulated tethered AUV environment (Figure 3). We place a single reward of 100 in each Ridge region, five steps to the right and five steps down from the AUV's initial deployment location. With a limit of 10 steps for each AUV deployment, this reward is very sparse and takes many rollouts to obtain. We place no reachable reward in the Caldera regions. We ran each of our algorithms to produce a reward-maximizing plan on a budget of 70 units of time. The optimal strategy is to visit only Ridge regions and obtain a reward of 600 via six successful deployments (the agents are allowed to go over the time budget on the last deployment).

In testing, Flat MCTS timed out without achieving any reward. Naive and Online Hierarchical approaches consistently achieved the maximum reward of 600. Note also that Flat MCTS was allowed to run for substantially longer than the other two algorithms. (Figure 2)

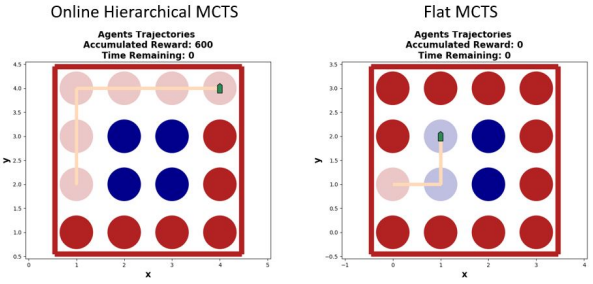

Figure 2. Left: Resulting motion of the mothership using Online Hierarchical MCTS. The maximum reward is obtained. Right: Resulting motion of the mothership using Flat MCTS. The AUV fails to collect reward in the first Ridge and the mothership strays into the Caldera. Flat MCTS was given a shorter time budget of 30 time units due to resource constraints.

7. Demonstration

We have included example tests for each algorithm in the src/hierarchical folder of our implementation. You can run hierarchical_example_flat.py to perform Flat MCTS, hierarchical_example_naive.py to perform Naive Hierarchical MCTS, and hierarchical_example_online.py to perform Online Hierarchical MCTS in our tethered AUV environment. The resulting plots represent the movement of the mothership in its grid, and the final reward collected by the AUV is indicated at the top.

8. Discussion

Our results indicate that both Naive and Online Hierarchical MCTS are able to significantly outperform Flat MCTS in our tethered-AUV domain. This is because both hierarchical methods are able to learn a macro-action which generalizes

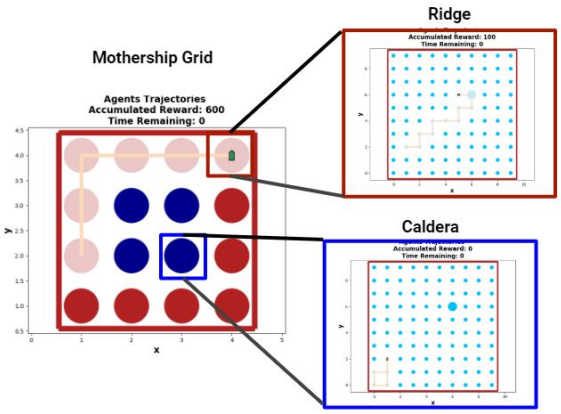

Figure 3. Visualized simulation of tethered AUV planning in an underwater volcanic environment. The mothership (green) makes large motions which may take it between Ridge-type (red) and Caldera-type (blue) regions of the environment. The tethered AUV (yellow) can achieve a sparse reward in the Ridge region but not the Caldera region.

across the domain to obtain the sparse reward in all Caldera regions. These methods are then able to generate a long-term plan for the mothership which executes that macro-action as much as possible. The difference between Online and Naive Hierarchical MCTS is not evaluated by our domain, because the number of states for the mothership is very small. Scaling up the size of our domain would allow for a comparison between these algorithms. Flat MCTS, on the other hand, does not exploit the generality of its AUV action-sequences and fails to collect the sparse reward on most deployments.

9. Conclusion

Hierarchical MCTS is a promising approach for asymmetric multi-agent collaboration, which necessitates both long-term and short-term planning. Hierarchical approaches can offer significant planning speedup in such scenarios when applied correctly.

Our evaluation of hierarchical MCTS was limited by several modeling simplifications. Future work should extend Hierarchical MCTS to more realistic tethered AUV environments where the true reward distribution is not known from the start and the agents must adaptively sample. This extension should be fairly straightforward, but the effectiveness of hierarchical MCTS in an adaptive sampling environment is unknown.

In the future, it would also be interesting to combine a hierarchical problem formulation with the decentralized planning which Zachary Duguid, my MCTS teammate, implemented as another extension to our project. Whereas the scenario presented in this paper is one of complete communication between agents, decentralized MCTS considers the scenario where communication between agents is limited. Working together to formulate our extensions, Zach and I considered an extension where agents could decide when to communicate, rather than assuming that communication occurs randomly or at a fixed interval. This scenario would lend itself well to hierarchical planning, because agents could develop plans over two macro-actions, ”explore” and ”communicate”, each with a differently defined reward function. Coordination of primitive actions and macro-actions under limited communication would be a further novel extension combining our two approaches.

10. Self-evaluation

Because our team divided up during the advanced lecture period of the class, with Jacob, Fillipos, and myself presenting the lecture while Zach and Can worked on the problem set, I did not play a major role in the original implementation of our core MCTS algorithm. However, I became very familiar with the code during the finalization of our problem set and at the start of our Grand Challenge push. This allowed me to contribute effectively to my team as we planned our approach for the Grand Challenge and formulated extensions. In the final weeks of class, I collaborated particularly closely with Zach to define each of our novel extensions. I then worked independently to flesh out and implement the Hierarchical MCTS extension. Overall, I believe that every member of our team made strong contributions to our final product.

References

Best, G. et al. Dec-mcts: Decentralized planning for multi-robot active perception. *The International Journal of Robotics Research*, 2018.

Kurzer, K., Zhou, C., and Zollner, J. Decentralized cooperative planning for automated vehicles with hierarchical monte carlo tree search. In *IEEE Intelligent Vehicles Symposium*, 2018.

Nguyen, K. and Thawonmas, R. Monte carlo tree search for collaboration control of ghosts in ms. pac-man. *IEEE Transactions on Computational Intelligence and AI in Games*, 5:57–68, 2013.

Vien, N. and Toussaint, M. Hierarchical monte-carlo planning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.