Stochastic Tree Search with Useful Cycles for Patrolling Problems

Bilal Kartal, Julio Godoy, Ioannis Karamouzas, and Stephen J. Guy

***Abstract*— An autonomous robot team can be employed for continuous and strategic coverage of arbitrary environments for different missions. In this work, we propose an anytime approach for creating multi-robot patrolling policies. Our approach involves a novel extension of Monte Carlo Tree Search (MCTS) to allow robots to have life-long, cyclic policies so as to provide continual coverage of an environment. Our proposed method can generate near-optimal policies for a team of robots for small environments in real-time (and in larger environments in under a minute). By incorporating additional planning heuristics we are able to plan coordinated patrolling paths for teams of several robots in large environments quickly on commodity hardware.**

I. Introduction

Multi-robot systems are nowadays commonly used to perform critical tasks, such as search and rescue operations, intelligent farming, mine sweeping and environmental monitoring [24]. In such tasks, robots have to observe or sweep an environment by solving a coverage planning problem. A variant of this problem is the multi-robot patrolling problem, in which multiple robots must coordinate their motions in order to minimize the probability of intrusion. Unlike coverage, patrolling needs to be performed continuously and in a strategic manner, given its adversarial nature.

The multi-robot patrolling problem has been an active research area within the last decade, especially as more and more autonomous robots are available for surveillance tasks at low costs. This problem is $\mathcal{NP}$-hard in its general sense [6]. Therefore, several heuristics and approximation algorithms have been proposed, with efficient solutions found for simple cases. In this paper, we adapt Monte Carlo Tree Search (MCTS) algorithm [12] to generate patrolling policies across arbitrary environments. MCTS can successfully search in large domains by using random sampling. The algorithm is anytime and converges to optimal solutions given enough time and memory for finite-horizon problems.

One of the main challenges to adapt MCTS to the patrolling domain is the ability to generate infinite length policies; the policies generated by MCTS are valid for a small time horizon while patrolling task has to be performed continuously. We address this issue by introducing Monte Carlo Tree Search with Useful Cycles, MCTS-UC, which augments standard MCTS with *cyclic nodes* to return infinite, cyclic policies.

This work makes three contributions. First, we propose the use of stochastic tree search for patrolling policy generation. Second, we show how useful cycles can be incorporated into MCTS to efficiently generate continuous cyclic policies without losing convergence guarantees. Finally, we experimentally show the applicability of MCTS-UC across a variety of scenarios. Coupled with a pruning heuristic, our approach can generate policies for intractably large environments.

*This work was supported in part by the University of Minnesota Supercomputing Institute

The authors are with the Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN, USA. {bilal,godoy,ioannis,sjguy}@cs.umn.edu

II. Related Work

In this section, we overview relevant work in the areas of multi-robot patrolling and MCTS. We refer the reader to the surveys presented in [22] and [5] for more details.

A. Multi-robot Patrolling Problem

The multi-robot patrolling problem has been studied since at least the work of Machado et al. [14] and a diversity of patrolling strategies have been theoretically analyzed in the following years [6], [19], [21]. Patrolling strategies for multiple robots can be computed either in a centralized manner where a central entity computes the policy for all the robots [4], or in a decentralized manner where each robot computes its own policy based on its local state [15]. In general, centralized approaches lead to more optimal policies than its decentralized counterparts, but are more computationally demanding as the policy space is exponential with respect to the number of robots in the environment. In this paper, we plan over the joint patroller space and convert the resulting centralized policy into individual policies that each of the robots should follow.

Theoretical bounds on penetration probabilities for different types of intruders have been analyzed in both line and perimeter environments [2]. A diversity of strategies for patrollers have been proposed to account for static and stochastic intruders [1], including multiple intruders performing coordinated attacks [23]. Game theoretic formulations of the patrolling problem have also been analyzed where interactions between patrollers and intruders are modeled as a leader-follower game [4], [18]. Previous work has also focused on extending the longevity of the patrolling task by replacing robots based on their battery life [10]. In a recent work [20], a trust model has been proposed such that poorly performing patrollers are identified and patrolling tasks are reassigned dynamically.

A closely related problem to patrolling is the pursuit-evasion problem which has been extensively studied for single and multiple pursuer agents, see [7] for a recent survey. In Section III-B, we demonstrate how our approach can be applied to the pursuit-evasion problem studied in [17].

B. Monte Carlo Tree Search

Monte Carlo Tree Search is a heuristic search algorithm that has been particularly recognized after its breakthrough performance in the game Go [8]. Other than board and video games [9], MCTS has been further employed for a variety of domains ranging from optimized planning [13] to computerized narrative generation [11]. In this work, we use MCTS following the approach [12] which employs UCB (Upper Confidence Bounds) technique to balance exploration vs. exploitation during planning and present an extension to MCTS for infinite horizon settings in patrolling problems.

III. MCTS for Multi-robot Patrolling

MCTS is an anytime sampling based method, and all it needs is a strategy evaluation function for complete policies. This simplicity of requirements motivated us to investigate MCTS for the patrolling problem. The approach can easily be extended to arbitrary patrol environments and different arbitrary scenarios, since it will search for the optimal strategy with random sampling in an anytime fashion.

A. Problem Formulation

In our problem setting, we are given n patroller robots, $r = \{r_1, \ldots, r_n\}$, that have to periodically cover an environment to guard it from intrusions. For simplicity, we model the environment as an undirected graph $G = (V, E)$, where the vertices V denote the patrol regions and the edges E represent the connectivity between these regions. We assume that time can be discretized, and elimination of an intruder is instantaneous. Initially, at $t = 0$, each robot is placed at some vertex of the graph. At each discrete time step, the possible actions for a robot are to move to a neighbor vertex in G or to stay still. Multiple robots are allowed to occupy the same vertex simultaneously.

We assume that an intruder q enters the environment at a specified time t_e, and it takes t_p time steps to complete a successful attack. As typically assumed in the literature (e.g. [1], [2]), the intruder has the same motion model as the patroller robots, but this can also be altered to different models easily. We further assume that the intruder can enter the environment from any vertex in V. Two intruder models are considered:

1) Dynamic: The intruder enters at a random vertex at the same time as the patroller robots, $t_e = 0$, and performs a random walk exploring the environment.
2) Stationary: The intruder enters at a random time $t_e \geq 0$, at a random vertex and spends t_p timesteps at the location completing the intrusion.

Given a patroller team r and a single intruder q following one of the above models, our goal is to find a *joint* policy π for the robots that maximizes the likelihood of capturing the intruder before the attack is successful. We evaluate any policy π using the following function:

$$R(\pi) = \begin{cases} 1 & \text{if } \exists r_i \in r \text{ s.t. } r_i(t) = q(t) \text{ and } t \leq t_e + t_p \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Algorithm 1: MCTS Algorithm.

```
Input  : Budget
Output: Policy
while Budget > 0 do
    Node ← ucbSelection(root) ;
    ω ← rollout(node);
    backpropagate(R(ω)) ;
    Budget = Budget−1 ;
end
```

where $r_i(t)$ denotes the location of r_i at time t and $q(t)$ is the intruder's location at t. From Eq. 1, we assume a robot can only sense an intruder on the vertices of G at discrete time steps. We assumed discrete sensing model since continuous sensing causes more power usage and noisier observations that might be caused by robot motor noise or camera image blur. Importantly, this assumption makes the problem more challenging in the presence of dynamic intruders; if the intruder moves past a patroller while it crosses an edge, the intruder will not be detected.

B. Policy Generation

In this section, we explain how the MCTS approach can be employed to the multi-robot patrolling problem in order to generate near-optimal finite length trajectories for the patrolling robots.

An overview of the MCTS is presented in Algorithm 1. The algorithm maintains a tree structure where each node represents the complete state of the world, s_t, consisting of the locations of the robots at a certain time t, with t also referring to the depth of the node in the tree. The root of the tree contains the initial state of the world that corresponds to the initial locations of robots. Each link in the tree represents one possible joint action set consisting of n actions, one per each robot. The MCTS algorithm proceeds by repeatedly adding one node at a time to the current tree until a given budget (e.g., number of simulations) is met. The newly added node represents the resulting world state after applying the corresponding joint action to the robot team. For each potential joint action set, we keep track of how many times we have tried it, and what its average evaluation score is.

MCTS generates policies through uniform random *roll-outs*. A simulated policy ω consists of a sequence of joint action sets:

$$\omega = \{a_1, a_2, \ldots a_z, \xi_1, \xi_2, \ldots \xi_x\}, \quad (2)$$

where each a_i refers to deterministic action sets obtained from the existing tree and each ξ_i refers to uniform random action sets. The task of random rollouts is to provide a probabilistic evaluation of incomplete deterministic policies. After evaluating ω using Eq. 1, the resulting $R(\omega)$ is used to update the average evaluation scores from the leaf node to the root node while incrementing their visit counts, in a process known as *back-propagation*.

Choosing which child node to expand, that is, choosing which action set to employ next, is an exploration-exploitation problem. We want to primarily choose action sets that have high average evaluation scores, but we also need to explore other possible actions in case the random rollout was unlucky and does not capture the true potential of that joint action set. The exploration-exploitation dilemma has been well studied in AI in the context of Multi-armed bandit problem. In this work, we chose to use the Upper Confidence Bounds (UCB) approach [3]. The overall process of combining MCTS with UCB is often referred to as Upper Confidence Trees (UCT), which has been proven to converge to optimal solutions at polynomial rate for finite-horizon problems [12]. Applied to our problem formulation, each node n chooses its child c with the largest value of $f(c)$:

$$f(c) = W(\pi_c) + \sqrt{\frac{2 \ln n_v}{c_v}} \tag{3}$$

where $W(.)$ denotes the average evaluation score obtained by Eq. 1, π_c is the parent's policy so far including node c, n_v is the total number of times the parent node n has been visited, and c_v is the total number of times that the action transitioning node n to node c has been previously tried.

If a node with unexplored child is reached ($c_v = 0$) a new node is created for one of the unexplored action sets. After the rollout and back-propagation steps, the selection step is started from the root node again. This way, the tree can grow in an uneven manner, biased towards good policies. After a budget of a fixed number of rollouts has been reached, we construct the final policy π by walking down the tree from the root node by recursively selecting action sets with highest visit counts until hitting a non-parent node.

To demonstrate the suitability of standard MCTS in policy generation, we apply it to the pursuit-evasion problem described by Noori et al. [17]. Here the task is to find an optimal pursuing strategy on a line (see Fig. 2(b)) for a modified random walker evader that always moves either left or right, where $t_e = 0$ and t_p varies. The authors employed an MDP analysis to show that their results are near optimal. For this problem, MCTS generates policies with competitive capture probabilities for a budget of 1M as shown in Table I. The resulting policies are approximately equal to the scores reported in the MDP analysis across all the authors' scenarios, slightly over-performing for smaller policy lengths and slightly under-performing for larger policy lengths. This near-optimality motivated our extensions to MCTS for the patrolling domain as explained in the following section.

	$t_p = 9$	$t_p = 14$	$t_p = 19$	$t_p = 24$	$t_p = 29$
Noori et al. [17]	0.297	0.429	0.564	0.710	0.803
MCTS	0.301	0.432	0.569	0.692	0.740

TABLE I

CAPTURE PROBABILITIES FOR A LINE WHERE $|V| = 21$.

Algorithm 2: Rollout with cylic action sets

```
Input: leafNode
if leafNode == CyclicArm then
    PerformCyclicActions() ;
else
    RandomRollout();
end
```

Algorithm 3: Backpropagation with cyclic arm creation

```
Input: leafNode
tempNode ← leafNode.getParent();
while tempNode != root do
    if tempNode == leafNode then
        leafNode.CreateCyclicSiblingArm();
    end
    tempNode ← tempNode.getParent();
    UpdateVisitCounts();
    UpdateWinrates();
end
```

IV. MONTE CARLO TREE SEARCH WITH USEFUL CYCLES (MCTS-UC)

Standard MCTS is limited to produce finite policies for finite search budgets. However, the patrolling problem requires the generation of infinite policies. To address this issue, we propose an MCTS variant, Monte Carlo Tree Search with Useful Cycles (MCTS-UC) that generates continuous cyclic policies for the patroller team. The notion of useful cycles has been previously studied by Nieuwenhuisen and Overmars [16] to improve path quality on probabilistic roadmaps. Following their work, we define a useful cycle as a set of patrolling paths that starts and ends in the same vertex set for all robots. Therefore, we exploit the spatial similarity of visited vertices of patrollers, i.e. whether the same set of vertices are visited between any two states or not, to determine a useful cycle. In terms of implementation, MCTS-UC creates artificial cyclic nodes which represent continuous policies. These nodes will be part of the tree search during exploration-exploitation. Our MCTS-UC algorithm is summarized in Algorithms 2-3.

A. Spatial Similarity for State Equivalence

To find a useful cycle and generate a cyclic node, our approach first needs to determine the spatial similarity of robot locations among different states. We assume that two states are equivalent if the same set of vertices are occupied by the patrollers in both states. We check for equivalent states during back-propagation step (see Algorithm 3).

Consider, for example, two equivalent nodes A and B as shown in Fig. 1(a). Given these equivalent nodes, a cyclic node, node C, is created as a sibling arm to node B and its cyclic parent node is set to node A as depicted in Fig. 1(b). Node C will be part of UCB selection phase just as any ordinary arm. When node B is selected after creating node

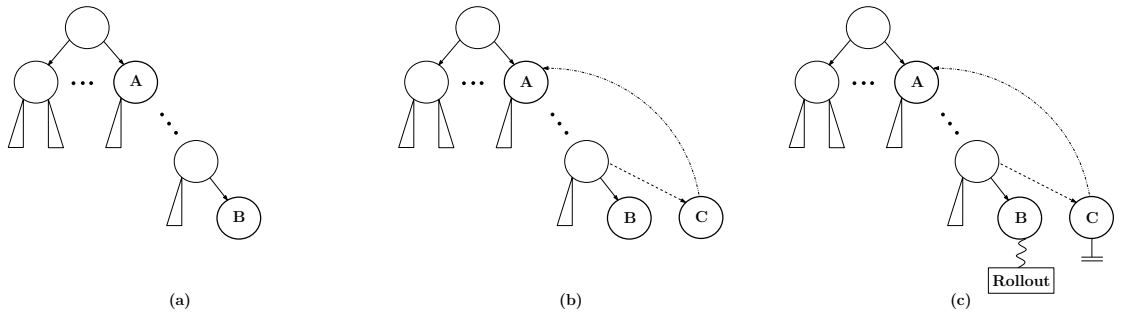

Fig. 1. Overview of Monte Carlo Tree Search with Useful Cycles: (a) During the back-propagation of node B, node A is found to have the same state. (b) A new cyclic node C is created to capture the cycle. (c) While node B is evaluated with standard rollouts, node C is evaluated cyclically.

C, we do a roll-out and expand the tree as in standard MCTS. However, if the cyclic node C is selected in further iterations of MCTS-UC, our algorithm creates a cyclic action buffer by pushing actions one by one while walking up the tree from itself to its cyclic parent (i.e. node A) and continuously performs these actions (Fig. 1(c)), evaluates the cyclic policy and back-propagates the policy score through the *non-cyclic* parent nodes as in standard MCTS. We should also note that both nodes A and B storing the equivalent states are kept unchanged to maintain the integrity of MCTS convergence conditions.

B. Mapping cycles to robots

We define a *cycle portion* $\psi = \{V_{begin}, \ldots, V_{end}\}$ of a patrol graph G as the set of vertices that transitions a robot from V_{begin} to V_{end}. For n robots, a *cycle* is comprised of n cycle portions between any equivalent two states. Let A_i denote the set of action sets corresponding to cycle portion ψ_i, $\forall i \in [1, n]$. Then, we can obtain a cyclic coverage in G if and only if all cycle portions, $\psi_1, \ldots, \psi_n$ are covered by some robot every $|A_i|$ time steps. Note that $|A_i| = |A_j|$ $\forall i, j$.

In order to generate a feasible cycle of actions for each robot, it is necessary to match each cycle portion to the corresponding robot every $|A_i|$ time steps. To find this matching, our MCTS-UC implementation maintains a list that keeps the first vertex V_{begin_i} of each cycle portion ψ_i. Let $p_i \in V$ denote the location of robot r_i. Then, we can assign robot r_i to ψ_j if and only if $p_i = V_{begin_j}$ and there is at least one V_{begin_j} in the list for any i, j. After the assignment we remove one instance of V_{begin_j} from the list and continue until the list is empty. If there exist multiple cycle portions with the same first vertex, we can assign the robot to any of the cycle portions arbitrarily as robots are assumed identical.

C. Convergence of MCTS-UC

MCTS-UC shares the convergence properties of the original MCTS for finite horizon problems. On one hand, the UCB equation (Eq. 3) guarantees that exploration always continues. On the other hand, the artificially added cyclic arms' rewards are also identically and independently distributed by Hoeffdings inequality as cyclic nodes are also evaluated for the same probabilistic intruder model. Similar to MCTS, the convergence analysis of MCTS-UC is also based on non-stationary arms having rewards sequences satisfying two drift conditions: 1) The expected average value of the arms has to converge to their true values $\overline{X}_{in} = \frac{1}{n}\sum_{k=1}^{n} X_{ik}$, where X_{ik} refers to the reward of i'th arm and k'th trial. From our policy evaluation function (Eq. 1) we know that $0 \leq X_{ik} \leq 1$ holds. Since we employ cyclic action sets repeatedly when cyclic nodes are selected, cyclic arms converge to their true values faster, empirically, as cyclic action sets lead to the coverage of the same regions repeatedly. 2) The tail distribution criteria should also be satisfied [12]. In MCTS-US this is true, since all cyclic and non-cyclic arms' rewards are identically and independently distributed.

We should note that as MCTS-UC has additional cyclic nodes at different levels of the tree, the required time to converge to the optimal policy might increase. For instance, if cyclic nodes close to the root node lead to sub-optimal but much better policies than that of sibling nodes, we might end up having longer convergence time.

D. Iterative MCTS-UC

Both MCTS and MCTS-UC generate an asymmetric tree as they grow the tree in regions where they estimate the existence of better solutions. However, for many problems including the multi-robot patrolling problem many symmetric solutions might exist that neither MCTS nor MCTS-UC are able to distinguish. For example, patrolling clockwise or counter-clockwise on a perimeter can be equally optimal according for a stationary intruder. Symmmetrical solutions causes scalability challenges and high amount of unnecessary exploration. Also, we have scalability issues while using MCTS or MCTS-UC standalone due to large team and environment sizes. For example, the branching factor can easily reach thousands in a grid graph where $n = 5$. To address these scalability issues, we employ a previously proposed *Iterative heuristic* [11] that scales very well and generates locally optimal solutions for scenarios that are intractable for standard MCTS methods. The Iterative MCTS-UC algorithm commits to the best evaluated action at every fixed iteration budget, and prunes all other nodes at that level. Selection of a cyclic node at the end of any iteration terminates the search and returns the corresponding cyclic policy.

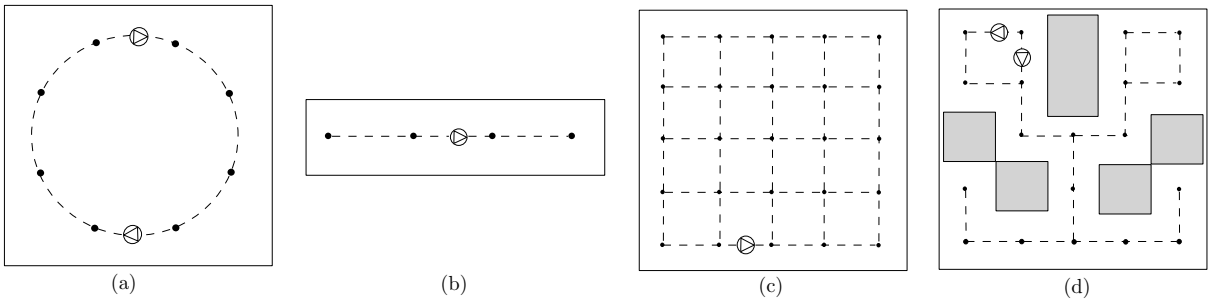

Fig. 2. Examples of patrol graphs used in our experiments for n robots:(a) A circular scenario where $|V| = 8$ and $n = 2$. (b) A line scenario where $|V| = 4$ and $n = 1$. (c) A grid scenario where $|V| = 25$ and $n = 1$. (d) A grid scenario with obstacles where $|V| = 19$ and $n = 2$.

Budget	1K	10K	500K	3M	5M	6M
MCTS	0.1778	0.231	0.2698	0.3012	0.3125	-
MCTS-UC	0.1612	0.171	0.255	0.3217	1	1

TABLE II

CAPTURE PROBABILITIES IN A PERIMETER WITH $|V| = 15$ AND $n = 3$.

V. RESULTS

We evaluated our MCTS-UC algorithm by comparing it to standard MCTS in terms of capture probability in three different environments: a line, a perimeter, and a 2D grid as shown in Fig. 2. We considered the two types of intruder strategies described in Section III-A.

A. MCTS vs MCTS-UC

As previously discussed, infinite length policies are required in the multi-robot patrolling domain. Unlike standard MCTS, our MCTS-UC approach can successfully generate such policies. Table II shows the capture probabilities of a stationary intruder on a perimeter (Fig. 2(a)), where the number of vertices $|V|$ is 15 and the number of patrollers n is 3. As it can be seen in Table II, for large simulation budgets MCTS-UC leads to policies with higher capture probabilities. However, for smaller budgets up to 500K, MCTS is slightly better than MCTS-UC. The reason behind this is that MCTS-UC invests some of its exploration budget on creating cyclic nodes, which pays off when a larger budget is available ($\geq$ 3M simulations), converging to an optimal cyclic policy given a budget of 5M simulations. In contrast, MCTS produces short-term policies for the same budget sizes, and is intractable after 5M explored nodes.

In a grid environment with $n = 2$ and a dynamic intruder (Fig. 2(c)) MCTS-UC outperforms MCTS for varying t_p's as shown in Table III. Note that the capture probability never reaches 1, as intruders that cross paths with patrollers (in edges) are not captured. In the same scenario with $n = 1$ against a stationary intruder, MCTS-UC creates policies with capture probability of 0.81, and 1 for $t_p = 25$, and $t_p = 50$ respectively.

B. Patrolling Strategies Evaluation

We have shown that MCTS-UC generates continuous policies where standard MCTS fails to do so. We also compare our generated policy with the optimal policy for a given environment. For perimeter patrolling, the optimality of cyclic strategies has been proven for single and multiple robots [6]. For multiple robots, optimality holds if the robots are placed equidistantly and they move in the same direction. Our approach, MCTS-UC, generates optimal cyclic policies even if all robots start to patrol from the same vertex. Fig. 3, for example, shows the trajectories of 2 robots patrolling on a perimeter. Starting from the same vertex, the robots move initially to opposite directions and place themselves equidistantly after 2 time steps, after which they patrol in the same direction indefinitely.

MCTS-UC can also generate partitions of the patrolling space in an emergent fashion. For example, Fig. 2(d) depicts a stationary intruder in a grid environment with obstacles and two patrollers. Our approach partitions the grid into two cycle portions covering the top and bottom regions with each robot patrolling its own partition. Videos can be seen at `http://motion.cs.umn.edu/r/MCTS-UC`

	$t_p = 30$	$t_p = 40$	$t_p = 50$
MCTS	0.532	0.551	0.592
MCTS-UC	0.626	0.774	0.829

TABLE III

CAPTURE PROBABILITIES IN A GRID ENVIRONMENT WITH $|V| = 25$ AND $n = 2$. BUDGET IS FIXED TO 1280K NODES.

C. Performance Analysis

1) Scalability: Besides having better capture probabilities than MCTS, MCTS-UC uses an order of magnitude less memory than MCTS. This is because whenever a useful cyclic node is selected, the search tree will not grow for that simulation as the cyclic action set will be performed repeatedly. However, computing policies for all robots can be prohibitively expensive for large environments. In these cases, heuristics that reduce the search space can mitigate the problem. For this purpose, we evaluated MCTS-UC with the Iterative heuristic on a 20x20 grid where $|V| = 400$, $n = 2$, and $t_p = 200$ for a dynamic intruder. We compare the capture probabilities of MCTS-UC with and without the Iterative heuristic. In this scenario, even a simple sweep of the environment requires 200 optimal moves which makes the problem intractable for MCTS-UC without the Iterative heuristic when using large budgets.

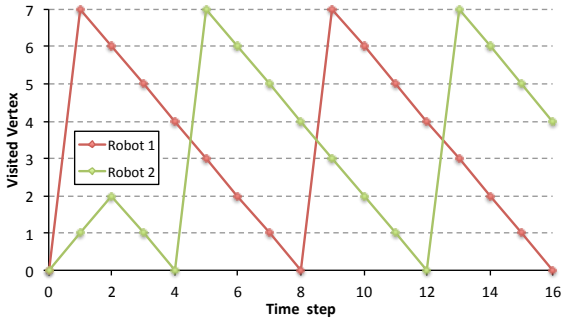

Fig. 3. Trajectories of 2 robots on a perimeter, $|V| = 8$. Both robots start at the same vertex and they adjust their placements to equidistant vertices and patrol in the same direction continuously. V_7 and V_0 are adjacent vertices.

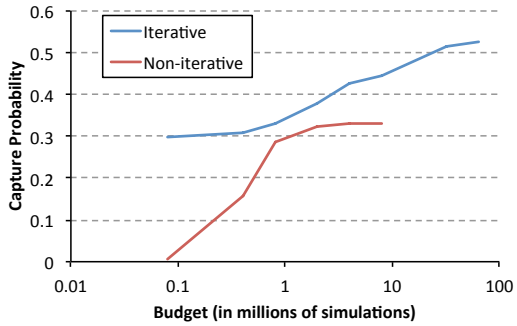

Fig. 4. Capture probabilities of Iterative MCTS-UC as compared to standard MCTS-UC (x-axis is log-scaled).

2) Runtime: The runtime of MCTS based approaches relies on the branching factor and the depth of the search tree. The Iterative heuristic provides runtime improvement in later simulations as it increases the number of initial committed steps. We can observe in Fig. 4 that Iterative MCTS-UC is able to find better policies with larger budgets in less time. For the perimeter scenario shown in Fig. 2(a), MCTS-UC requires 10 seconds to obtain an optimal policy, and the grid scenario shown in Fig. 2(b) requires one minute for an optimal solution with MCTS-UC.

VI. Conclusion

We have presented an anytime approach capable of generating continuous policies for patrolling problems with different intruder models. Our approach exploits spatial similarity of different robot configurations with time shifts to create infinitely long policies from finite simulation budget. It generates theoretically proven optimal policies for perimeters and near-optimal policies for arbitrary environments. One limitation of using MCTS-UC in our domain is the high branching factor due to joint-action space. While its spatial complexity scales polynomially with respect to the environment size, the degree of each node scales exponentially with respect to the number of robots, e.g. grid environments with $n \geq 5$ are intractable for MCTS-UC as the branching factor exceeds one thousand. To alleviate this challenge, we will explore MCTS parallelization techniques. Lastly, we plan to investigate more complex classes of patrolling problems by prioritizing target points, or by limiting intruders to specific entry and exit points.

References

[1] N. Agmon, G. A. Kaminka, and S. Kraus. Multi-robot adversarial patrolling: Facing a full-knowledge opponent. *Journal of Artificial Intelligence Research*, 42:887–916, 2011.

[2] N. Agmon, S. Kraus, and G. A. Kaminka. Multi-robot perimeter patrol in adversarial settings. In *IEEE International Conference on Robotics and Automation*, pages 2339–2345, 2008.

[3] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[4] N. Basilico, N. Gatti, and F. Villa. Asynchronous multi-robot patrolling against intrusions in arbitrary topologies. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[5] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *IEEE Trans. on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[6] Y. Chevaleyre. Theoretical analysis of the multi-agent patrolling problem. In *Proceedings of Intelligent Agent Technology (IAT)*, 2004.

[7] T. H. Chung, G. A. Hollinger, and V. Isler. Search and pursuit-evasion in mobile robotics. *Autonomous Robots*, 31(4):299–316, 2011.

[8] M. Enzenberger, M. Muller, B. Arneson, and R. Segal. Fuego—an open-source framework for board games and go engine based on monte carlo tree search. *IEEE Trans. on Computational Intelligence and AI in Games*, 2(4):259–270, 2010.

[9] E. J. Jacobsen, R. Greve, and J. Togelius. Monte mario: platforming with MCTS. In *Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 293–300, 2014.

[10] E. Jensen, M. Franklin, S. Lahr, and M. Gini. Sustainable multi-robot patrol of an open polyline. In *IEEE International Conference on Robotics and Automation*, pages 4792–4797, 2011.

[11] B. Kartal, J. Koenig, and S. J. Guy. User-driven narrative variation in large story domains using monte carlo tree search. In *Autonomous agents and multi-agent systems*, pages 69–76, 2014.

[12] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, pages 282–293. Springer, 2006.

[13] M. Levihn, J. Scholz, and M. Stilman. Hierarchical decision theoretic planning for navigation among movable obstacles. In *Algorithmic Foundations of Robotics X*, pages 19–35. Springer, 2013.

[14] A. Machado, G. Ramalho, J.-D. Zucker, and A. Drogoul. Multi-agent patrolling: An empirical analysis of alternative architectures. In *Multi-Agent-Based Simulation II*, pages 155–170. Springer, 2003.

[15] A. Marino, L. Parker, G. Antonelli, and F. Caccavale. Behavioral control for multi-robot perimeter patrol: A finite state automata approach. In *IEEE International Conference on Robotics and Automation*, pages 831–836, 2009.

[16] D. Nieuwenhuisen and M. H. Overmars. Useful cycles in probabilistic roadmap graphs. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 446–452, 2004.

[17] N. Noori, A. Renzaglia, and V. Isler. Searching for a one-dimensional random walker: Deterministic strategies with a time budget when crossing is allowed. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4811–4816, 2013.

[18] P. Paruchuri, J. P. Pearce, M. Tambe, F. Ordonez, and S. Kraus. An efficient heuristic approach for security against multiple adversaries. In *Autonomous agents and multiagent systems*, page 181, 2007.

[19] F. Pasqualetti, A. Franchi, and F. Bullo. On cooperative patrolling: Optimal trajectories, complexity analysis, and approximation algorithms. *Robotics, IEEE Transactions on*, 28(3):592–606, 2012.

[20] C. Pippin and H. Christensen. Trust modeling in multi-robot patrolling. In *IEEE International Conference on Robotics and Automation*, pages 59–66, 2014.

[21] D. Portugal, C. Pippin, R. P. Rocha, and H. Christensen. Finding optimal routes for multi-robot patrolling in generic graphs. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 363–369. IEEE, 2014.

[22] D. Portugal and R. Rocha. A survey on multi-robot patrolling algorithms. In *Technological Innovation for Sustainability*, pages 139–146. Springer, 2011.

[23] E. Sless, N. Agmon, and S. Kraus. Multi-robot adversarial patrolling: facing coordinated attacks. In *Autonomous agents and multi-agent systems*, pages 1093–1100, 2014.

[24] J. Vander Hook, P. Tokekar, E. Branson, P. G. Bajer, P. W. Sorensen, and V. Isler. Local-search strategy for active localization of multiple invasive fish. In *Experimental Robotics*, pages 859–873, 2013.