2024 IEEE International Conference on Robotics and Automation (ICRA)
May 13-17, 2024. Yokohama, Japan

2024 IEEE International Conference on Robotics and Automation (ICRA) | 979-8-3503-8457-4/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICRA57147.2024.10610888

Multi-Level Action Tree Rollout (MLAT-R): Efficient and Accurate Online Multiagent Policy Improvement

Andrea Henshall[1] and Sertac Karaman[2]

***Abstract*— Rollout algorithms are renowned for their abilities to correct for the suboptimalities of offline-trained base policies. In the multiagent setting, performing online rollout can require an exponentially large number of optimizations with respect to the number of agents. One-agent-at-a-time algorithms offer computationally efficient approaches to guaranteed policy improvement; however, this improvement is with respect to a state value estimate derived from a potentially poor base policy. Monte Carlo tree search (MCTS) provably converges to the true state value estimates; however, the exponentially large search space often makes its online use limited. Here, we present the Multi-Level Action Tree Rollout (MLAT-R) algorithm. MLAT-R provides 1) provable improvement over a base policy, 2) policy improvement with respect to the true state value, 3) applicability to any number of agents, and 4) an action space that grows linearly with the number of agents rather than exponentially. In this paper, we outline the algorithm, sketch a proof of its improvement over a base policy, and evaluate its performance on a challenging problem for which the base policy cannot reach a terminal state. Despite the challenging experimental setup, our algorithm reached a terminal state in 86% of all experiments, compared to 31% for state-of-the-art one-agent-at-a-time algorithms. In experiments involving MCTS, MLAT-R reached a terminal state in 99% of experiments compared to 92% for MCTS. MLAT-R achieved these results while considering an exponentially smaller action space than MCTS.**

I. INTRODUCTION

Multiagent, sequential decision-making problems can be found in a wide variety of applications including vehicle routing [1], task allocation [2], and search and rescue [3] [4]. While using multiple agents to achieve a joint goal can reduce the number of steps required to do so, finding exact solutions to such problems in real-time, especially in a changing environment, is often intractable. As the size of the joint action space in multiagent problems can increase exponentially with the number of agents, policies are often approximated using offline training. These "base" policies, however, are often suboptimal for a variety of reasons including insufficiently rich policy architecture, insufficient or non-representative training data, and difficulties training a policy to a global optimum in a complex reward landscape. While online "rollout" policies using one- or multi-step lookahead [5] [6] [7] [8] [9] [10] [11] and Monte Carlo tree search (MCTS) [12] [13] [14] have been used with great success to correct for some of the base policy's suboptimalities, they have drawbacks. Some versions of the one-agent-at-a-time rollout algorithm have been shown to have the policy improvement property with an action space that increases in size linearly with the number of agents. This improvement, however, is relative to the state value derived from the base policy. If the base policy is poor, this approach may not offer improvement with respect to the true state value. Conversely, MCTS provably converges to the true state value; however, its use for online policy improvement can be precluded by the potentially exponentially large joint action space considered at each level in the tree.

Here we present the Multi-Level Action Tree Rollout (MLAT-R) algorithm, which uses Monte Carlo methods to build a multi-level action tree (MLAT), which when traversed, produces an improved policy. Our work offers the following contributions: 1) provable improvement over a base policy, 2) improvement with respect to the true state value, 3) applicability to any number of agents, and 4) an exponential decrease in the number of actions considered at each level over MCTS.

The rest of this paper is organized as follows: we describe the mathematical background for our algorithm and related work in section II, outline our algorithm in section III, provide a proof sketch of the policy improvement property in section IV, compare our algorithm with state-of-the-art one-at-a-time algorithms in section V, and outline our conclusions and future work in section VI.

II. BACKGROUND AND RELATED WORK

Offline-trained, multiagent policies may be suboptimal for a variety of reasons previously outlined. The "rollout algorithm" is a conceptually simple way of correcting for a base policy's suboptimalities. The algorithm's name was originally coined by G. Tesauro, who used a truncated version of it to play backgammon at an expert level [5] [6]. Tesauro describes a single-agent scenario for which a_k is the action taken at step k out of a possible set of actions, A_k. The reward for taking action a_k at state s_k is $r(s_k, a_k)$, and we define $V_{k,\pi}$ as the state value when following policy, π. We define f_k as the function which maps s_k and a_k to s_{k+1}. To keep the total reward bounded over a trajectory, we use a discount term, $\gamma \in (0, 1]$. A single-agent, single-step lookahead rollout algorithm then takes a base policy from time step 0 to time step $T-1$, $\pi = \{\mu_0, \mu_1, ..., \mu_{T-1}\}$, and at each state, s_k, replaces the policy μ_k with $\tilde{a}_k$ such that:

$$\tilde{a}_k \in \underset{a_k \in A_k}{\arg\max}\ E\Big(r\big(s_k, a_k\big) + \gamma V_{k+1,\pi}\big(f_k(s_k, a_k)\big)\Big) \quad (1)$$

[1]Andrea Henshall is with the Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, 77 Massachusetts Ave, Cambridge, MA, 02139, USA amhenshall@mit.edu

[2]Sertac Karaman is with the Faculty of Aeronautics and Astronautics, Massachusetts Institute of Technology, 77 Massachusetts Ave, Cambridge, MA, 02139, USA sertac@mit.edu

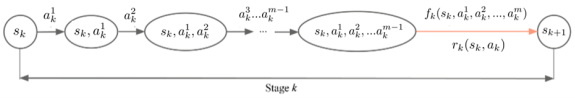

Fig. 1: Modified State Transition for Sequential Multiagent Lookahead [8]

The rollout algorithm possesses the reward improvement property: it is guaranteed to perform at least as well as the base policy. This property extends to multi-step lookahead with improved performance expected at the cost of increased computation time. Even with a relatively poor base policy, the rollout algorithm has been shown to offer improvement [8].

The complexity of the rollout algorithm described above is related to the size of the action space rather than the state space as can be seen in (1). For a multiagent scenario, the size of the joint action space can grow exponentially if the joint space can be represented as the Cartesian product of the individual agent action spaces: $A_k(s_k) = A_k^1(s_k) \times A_k^2(s_k) \times ... \times A_k^m(s_k)$ with $a_k^m(s_k) \in A_k^m(s_k)$ and $a_k(s_k) = (a_k^1(s_k), a_k^2(s_k), ..., a_k^m(s_k))$ for m number of agents. If $|A|$ is the upper bound on the number of actions available to an agent at any time step, there are as many as $|A|^m$ elements over which to perform the minimization in (1).

Because the complexity of the rollout algorithm is not dependent on the size of the state space, D. Bertsekas sought to trade action complexity for state complexity. He introduced intermediate states represented by the tuple of the state at step k and combinations of individual agent actions at step k [15]. Transitions from state s_k to the intermediate states to state s_{k+1} can be seen in Fig. 1. As this representation is identical to the original problem, the optimal joint action for the modified representation is also the optimal joint action for the original multiagent problem. An approach to solving this reformulated version is to simply use Tesuro's rollout algorithm from s_k to (s_k, a_k^1) to (s_k, a_k^1, a_k^2) through all $m-1$ intermediate states to s_{k+1}. Assuming the base policy at step k is $\pi_k = \{\mu_k^1, \mu_k^2, ..., \mu_k^m\}$, and the updated policy using the rollout algorithm is $\tilde{\pi}_k = \{\tilde{\mu}_k^1, \tilde{\mu}_k^2, ..., \tilde{\mu}_k^m\}$, we determine $\tilde{\mu}_k^l$ as follows:

$$\tilde{\mu}_k^1(s_k) \in \underset{a_k^1 \in A_k^1(s_k)}{\arg\max} E\Big(r\big(s_k, a_k^1, \mu_k^2(s_k), ..., \mu_k^m(s_k)\big) + \gamma V_{k+1,\pi}\big(f_k(s_k, a_k^1, \mu_k^2(s_k), ..., \mu_k^m(s_k))\big)\Big) \quad (2)$$

$$\tilde{\mu}_k^2(s_k) \in \underset{a_k^2 \in A_k^2(s_k)}{\arg\max} E\Big(r\big(s_k, \tilde{\mu}_k^1, a_k^2(s_k), \mu_k^3, ..., \mu_k^m(s_k)\big) + \gamma V_{k+1,\pi}\big(f_k(s_k, \tilde{\mu}_k^1, a_k^2, \mu_k^3(s_k), ..., \mu_k^m(s_k))\big)\Big) \quad (3)$$

...

$$\tilde{\mu}_k^m(s_k) \in \underset{a_k^m \in A_k^m(s_k)}{\arg\max} E\Big(r\big(s_k, \tilde{\mu}_k^1, \tilde{\mu}_k^2, ..., a_k^m(s_k)\big) + \gamma V_{k+1,\pi}\big(f_k(s_k, \tilde{\mu}_k^1, \tilde{\mu}_k^2(s_k), ..., a_k^m(s_k))\big)\Big) \quad (4)$$

We use the base policy for "future" agent actions between intermediate states and use previously determined actions from (2) through (4) for "past" agents. While the resulting joint action of this rollout algorithm may not produce the optimal joint action, one-at-a-time rollout possesses the reward improvement property, which Bertsekas proves using the Bellman equation in [8] [9].

At each step, the rollout algorithm optimizes an action with respect to a state value drawn from what may be a poor policy. Multiple approaches have been taken to improve the state value estimate. [16] corrects for base policy suboptimalities due to distribution shifts between training and observed data by training multiple base policies on various distributions. The authors demonstrate an improved rollout policy over one-at-a-time rollout with a fixed base policy when dispatching taxis responding to ride requests. Unfortunately, it can be challenging to parameterize potentially complex training data distributions and identify when observed data moves outside of a given distribution and into another. [17] attempts to improve the state value estimate online using the rewards from various mixed integer programming (MIP) solutions and known probabilities of task failure when allocating tasks to human/machine teams. This approach, however, can be time-consuming for online applications and requires knowledge of state transition probabilities, which may be unrealistic for some problems. Finally, Monte Carlo tree search (MCTS) has been shown to converge to the true state values with enough trajectory simulations [18]. While this approach has been successfully used to achieve superhuman performance in games like Go, Chess, Shogi, Tron, and more [12] [13] [19] [20], the large multiagent action space makes it too inefficient for time-sensitive online use. [21] describes a cooperative multiagent MCTS approach with a pruned action space at each level; however, their demonstrations are limited to simple two-player matrix games, and while the results are promising for these demonstrations, the authors offer no proof the approach provides the policy improvement guarantee we seek.

III. ALGORITHM

We model our system as a Markov Decision Process (MDP) with the joint state fully observable by all agents. We permit unrestricted communication between agents and assume we have access to a base policy and some means of estimating the value of a joint state. We do not restrict the type of policy nor means of estimating a state value (heuristic, function approximation, etc.), and it need not even be very "good." The central part of our algorithm is the MLAT, which at each level, contains representations of the states shown in Fig. 1. A pictorial representation of the MLAT can be seen in Fig. 2. Algorithm 1 outlines

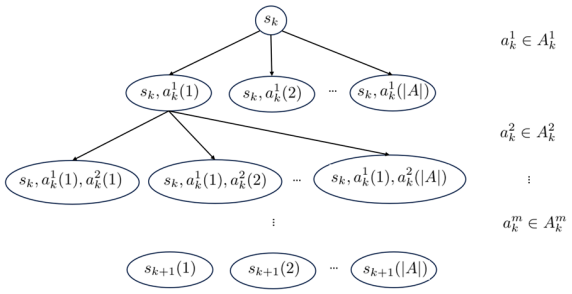

Fig. 2: Multi-Level Action Tree (MLAT)

the construction of the MLAT. n_{root} is the root node, s_n is the state of node n, $N_{\text{sims}}^{\max}$ is the maximum number of simulations, C_n is the set of all children of node n, and N_{visits}^n is the number of visits to node n. R_n is an estimate of the cumulative reward of moving to n from its parent then along some trajectory to a terminal state. UBC is the upper confidence bound, from which there are several versions to pick (we discuss our choice later). Rollout (line 17) can be accomplished in a variety of ways depending on the accessibility of the MDP (to include taking random actions). Here, we use the action value estimate for node n as $R_n = r(s,a) + \gamma V_\pi\big(f(s,a)\big)$.

Algorithm 1 Multi-Level Action Tree Construction

1: Input: n_{root}, dynamics model or simulator, $N_{\text{sims}}^{\max}$
2: Output: Multi-Level Action Tree
3: **if** s_{root} is terminal **then**
4: **return** n_{root}
5: **end if**
6: EXPAND(n_{root})
7: $N_{\text{sims}} \leftarrow 0$
8: **while** $N_{\text{sims}} < N_{\text{sims}}^{\max}$ **do**
9: $n \leftarrow n_{\text{root}}$
10: **while** n is expanded **do** ▷ Until leaf node
11: $n \leftarrow \arg\max_{c \in C_n} UCB(c)$ ▷ Pick best child
12: **end while**
13: **if** $N_{\text{visits}}^n \neq 0$ and $s_n \neq s_{\text{terminal}}$ **then**
14: EXPAND(n)
15: $n \leftarrow \arg\max_{c \in C_n} UCB(c)$
16: **end if**
17: $R_n \leftarrow$ ROLLOUT(n) ▷ Sim. trajectory reward
18: BACKPROPAGATE(R_n)
19: $N_{\text{sims}} \leftarrow N_{\text{sims}} + 1$
20: **end while**

Each time a leaf node (unexpanded node at the end of a branch of the tree) is expanded (lines 6 and 14), only the actions available to that agent are considered, so the number of children is equal to $|A_k^l|$, the number of actions available to agent l at step k. This is potentially an exponential reduction in the number of children compared to using the joint action space ($|A|$ versus $|A|^m$). Algorithm 2 shows the procedure for expanding a leaf node. n_k^l is the node for agent l at time step k. These labels alone cannot identify a single node as there are multiple nodes for agent l at time k with different action histories (i.e. on different branches from the root node, but at the same depth in the tree). We omit these additional identifiers as they are not useful for understanding the algorithm. Depending on the formulation of the MDP, the set of available actions to agent l at time k, A_k^l, may depend on the actions taken by agents 1 through $l-1$. Such logic is easy to implement, so we omit it here for brevity. C_n is the set of all children to node n, a_c is the joint action for this child node with the actions for agents $1...l-1$ taken from the MCTS tree traversal policy, $\hat{\pi}$. Subsequent actions are taken from the base policy, π, as previously described. g_k is the reward function, which may be unknown, but from which we can receive output. f_k is the state transition function, which may also be unknown, but from which we can also receive output. We use γ as a discount factor, but for finite horizon and other bounded reward problems, it can be set to 1. The state of the node is either a true state or intermediate state as previously described and the priors, $P(s,a)$, for each node are generated from the softmax, σ, of the set of state action values for each child.

Algorithm 2 Expand

1: Input: n_k^l, set of legal actions A_k^l, dynamics model or simulator
2: Output: None
3: $C_n \leftarrow \{\}$
4: **for** $a \in A_k^l$ **do**
5: $a_c \leftarrow \{\{\hat{a}_k^{1...l-1}\}, a, \{\mu_k^{l+1...m}\}\}$
6: $r(s_n, a_c) \leftarrow g_k(s_n, a_c)$
7: **if** $l = m$ **then** ▷ Last element of the joint action
8: $s_c \leftarrow f_k(s_n, a_c)$
9: **else**
10: $s_c \leftarrow \{s_n, \{\hat{a}_k^{1...l-1}\}\}$
11: **end if**
12: $P(s_n, a_c) \leftarrow \sigma(r(s_n, a_c) + \gamma V_\pi(f_k(s_n, a_c)))$
13: $c_a \leftarrow \text{Node}(s_c, r(s_n, a_c), P(s_n, a_c), a_c)$
14: $C_n \leftarrow c_a \cup C_n$
15: **end for**

When backpropagating the reward received from rolling out a leaf state to a terminal state, care must be taken to backpropagate the correct value. As the transition between time steps k and $k-1$ (as we go up the tree) can pass through several intermediate nodes, the discount factor should be applied again only at a true state. This is indicated in Algorithm 3 for which l is a leaf node, Q_n is the running sum of all the rewards backpropagated through node n, and $\bar{Q}_n$ is the mean reward at node n.

As previous mentioned, we have multiple options for our UCB. Building off of [22], [23], [24], [25], and [26], [18] established the polynomial concentration property of regret for a class of non-stationary, multi-arm bandits, which proved that MCTS with the appropriate polynomial UCB exploration term converged to the value function for a given state with

Algorithm 3 Backpropagate

Input: n_k^l, r
Output: None
for $n \in$ path from n_k^l to n_{root} **do**
    $Q_n \leftarrow Q_n + r$
    $N^n_{\text{visits}} \leftarrow N^n_{\text{visits}} + 1$
    $\bar{Q}_n \leftarrow Q_n / N^n_{\text{visits}}$
    if Parent(n) is a true state **then**
        $r \leftarrow r(s_{k-1}, a_{k-1}) + \gamma \bar{Q}_n$
    end if
    $n \leftarrow$ Parent(n)
end for

enough simulations. For this reason, an exploration term proportional to the root of the number of visits to the parent node is used rather than its logarithm as in UCBs like UCB1 [27]. We use the same polynomial upper confidence tree (pUCT) UCB as [13] and the same exploration constant: $c_{puct} = 1$.

$$\text{pUCT}(n) = Q'(s_n, a_n) + c_{\text{puct}} P(s_n, a_n) \frac{\sqrt{N^{\text{parent}}_{\text{visits}}}}{1 + N^n_{\text{visits}}} \quad (5)$$

for which

$$Q'(s_k, a_k) = \frac{\bar{Q}(s_k, a_k) - \min\limits_{s,a \in \text{MLAT}} \bar{Q}(s, a)}{\max\limits_{s,a \in \text{MLAT}} \bar{Q}(s, a) - \min\limits_{s,a \in \text{MLAT}} \bar{Q}(s, a)} \quad (6)$$

We normalize our state action values based on those previously observed as in [14], so there is no need to know the reward bounds ahead of time.

IV. PROOF SKETCH OF POLICY IMPROVEMENT

To demonstrate our algorithm possesses the policy improvement property, we start by showing our MLAT will estimate the correct state value for intermediate and true states. For simplicity, we look at a two-agent system for which $|A| = 2$. Looking at the value backpropagated from an intermediate state to a true state, we have:

$$\bar{V}^i(s_k) = \frac{1}{N^{n(s_k)}_{\text{visits}}} \left(\sum_{d=1}^{N^{n(s_k, a_k^1 = 1)}_{\text{visits}}} r(s_k, 1, \mu_k^2) + \gamma \bar{V}_d(f_k(s_k, 1, \mu_k^2)) + \sum_{e=1}^{N^{n(s_k, a_k^1 = 2)}_{\text{visits}}} r(s_k, 2, \mu_k^2) + \gamma \bar{V}_e(f_k(s_k, 2, \mu_k^2)) \right) \quad (7)$$

for which $\bar{Q}(s_k, a_k) = r(s_k, a_k) + \gamma \bar{V}(f_k(s_k, a_k))$. We use $\bar{V}^i$ to indicate this is the contribution to $\bar{V}$ from an intermediate node. $\bar{V}_d$ indicates our value estimate is coming from the MLAT at visit d. We omit the time step subscript in favor of a visit identifier to emphasize the fact that our value estimate may change with each node visit. Equation (7) is the weighted (by the MCTS policy) average of the rewards over agent 1's actions given agent 2 will follow the base policy. Backpropagating to an intermediate state from two true states, we have:

$$\bar{V}^t(s_k, a_k^1) = \frac{1}{N^{(s_k, a_k^1)}_{\text{visits}}} \left(\sum_{d=1}^{N^{n(s_{k+1} | a_k^2 = 1)}_{\text{visits}}} r(s_k, \hat{\mu}_k^1, 1) + \gamma \bar{V}_d(s_{k+1} = f_k(s_k, \hat{\mu}_k^1, 1)) + \sum_{e=1}^{N^{n(s_{k+1} | a_k^2 = 2)}_{\text{visits}}} r(s_k, \hat{\mu}_k^1, 2) + \gamma \bar{V}_e(s_{k+1} = f_k(s_k, \hat{\mu}_k^2, 2)) \right) \quad (8)$$

for which $\hat{\mu}$ denotes the action was drawn from the MCTS policy. This is clearly the weighted (by the MCTS policy) average of the rewards over agent 2's actions given agent 1's actions. The backpropagation from s_{k+1} to s_k is:

$$\bar{V}(s_k) = \frac{1}{N^{n(s_k)}_{\text{visits}}} \left(\sum_{d=1}^{N^{n(a_k = \{1,1\})}_{\text{visits}}} r(s_k, \{1,1\}) + \gamma \bar{V}_d(s_{k+1} = f_k(s_k, \{1,1\})) + \sum_{e=1}^{N^{n(a_k = \{1,2\})}_{\text{visits}}} r(s_k, \{1,2\}) + \gamma \bar{V}_e(s_{k+1} = f_k(s_k, \{1,2\})) + \sum_{f=1}^{N^{n(a_k = \{2,1\})}_{\text{visits}}} r(s_k, \{2,1\}) + \gamma \bar{V}_f(s_{k+1} = f_k(s_k, \{2,1\})) + \sum_{g=1}^{N^{n(a_k = \{2,2\})}_{\text{visits}}} r(s_k, \{2,2\}) + \gamma \bar{V}_g(s_{k+1} = f_k(s_k, \{2,2\})) \right) \quad (9)$$

It can be seen from these equations that $\bar{V}(s_k)$ becomes a weighted average of the reward received by moving from state s_k to various s_{k+1} using their respective joint actions summed with the discounted estimate of the reward to continue on to the terminal node. $\bar{V}(s_k, a_k^1)$ is a weighted average of the rewards received by taking various a_k^1, but always μ_k^2. These are the values we seek. As μ is drawn from our base policy and our a_k is selected from our MCTS policy, our $\bar{V}(s_k)$ is some combination of the true state value and the state value from our base policy: $\bar{V}(s_k) = (1 - \beta) V_\pi(s_k) + \beta V(s_k)$ for which $\beta \in [0, 1]$. We know from [18] that $\beta \to 1$ as the number of simulations goes to infinity, so $\bar{V}(s_k) \to V(s_k)$. Once our MLAT is built, we select our actions one agent at a time as in [8], but with V_π replaced by V. Generalizing for an m-agent system again, traversing our MLAT is equivalent to solving the following equations:

$$\tilde{\mu}_k^1(s_k) \in \underset{a_k^1 \in A_k^1(s_k)}{\arg\max} E\Big(r\big(s_k, a_k^1, \mu_k^2(s_k), ..., \mu_k^m(s_k)\big) + \gamma V_{k+1}\big(f_k(s_k, a_k^1, \mu_k^2(s_k), ..., \mu_k^m(s_k))\big)\Big) \quad (10)$$

$$\tilde{\mu}_k^2(s_k) \in \underset{a_k^2 \in A_k^2(s_k)}{\arg\max} E\Big(r\big(s_k, \tilde{\mu}_k^1, a_k^2(s_k), \mu_k^3, ..., \mu_k^m(s_k)\big) + \gamma V_{k+1}\big(f_k(s_k, \tilde{\mu}_k^1, a_k^2, \mu_k^3(s_k), ..., \mu_k^m(s_k))\big)\Big) \quad (11)$$

...

$$\tilde{\mu}_k^m(s_k) \in \underset{a_k^m \in A_k^m(s_k)}{\arg\max} E\Big(r\big(s_k, \tilde{\mu}_k^1, \tilde{\mu}_k^2, ..., a_k^m(s_k)\big) + \gamma V_{k+1}\big(f_k(s_k, \tilde{\mu}_k^1, \tilde{\mu}_k^2(s_k), ..., a_k^m(s_k))\big)\Big) \quad (12)$$

We, therefore, asymptotically optimize our actions with respect to the true state value, offering further policy improvement over (2) through (4).

V. EXPERIMENTS AND RESULTS

We tested our algorithm against a scenario for which it was easy to see the optimal trajectory, but which required considerable coordination between agents: a multiagent shortest path problem. On a square grid, a specified number of agents started at distinct nodes near the lower left corner. A set of terminal nodes, which is a bijection of the set of agents, was designated on the top, right corner of the grid. This scenario is shown in Fig. 3. A terminal state was considered reached and a reward received when each terminal node was occupied by exactly one agent (we did not specify which agent must occupy which terminal node). Legal actions were up, down, left, and right to neighboring nodes and stay at the current node. All five actions incurred an equal cost unless the agent was at one of the designated terminal nodes. At a terminal node, the cost of staying at the node was 0. Agents were penalized for using the same node at the same time. The base policy was a deterministic, decentralized one in which each agent moved greedily toward the top, right corner of the grid, favoring moves to the right. This meant the base policy would drive all agents to the same node and they would never reach a terminal state. While such a problem would normally be solved using more straight-forward approaches [7] [28] [29], we used it here with an intentionally poor base policy to demonstrate the effectiveness of our algorithm to efficiently perform online policy correction.

We compared our algorithm to three other rollout algorithms: one-at-a-time, order-optimized, and MCTS. The one-at-a-time algorithm optimized each agent's actions sequentially as shown in (2) through (4). If multiple actions had the same value, an action was chosen randomly from among them. The order-optimized algorithm is the one-at-a-time algorithm further optimized for agent order. These algorithms are described in more detail in [8]. MCTS used the full joint action space to construct a tree given a specified number of

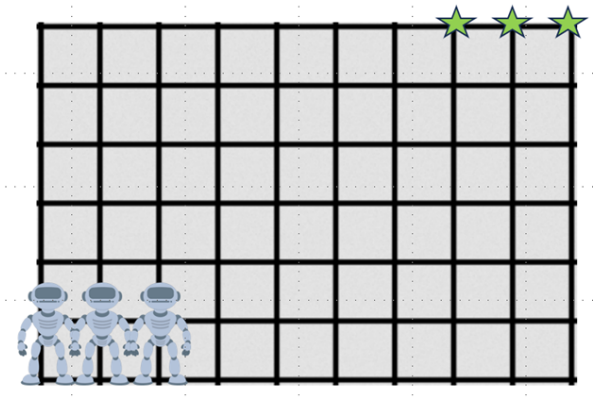

Fig. 3: Experimental Setup

TABLE I: Multiagent Shortest Path Experiment Parameters

Parameter	Value(s)
c_{pUCT}	1
$N_{\text{sims}}^{\max}$	100 - 400
Dirichlet Noise Fraction	0.25
Dirichlet α	10/(no. of legal actions from node)
Best Action Metric (Online)	Mean Value
Length of Square Grid	3-15 Units
Number of Agents	3-5
Movement Cost	1
Terminal Reward	2*(grid length)
Collision Cost	2*(grid length)
γ	0.99

simulations. It then traversed the tree using a given metric for determining the best child node. MLAT-R constructed an MLAT using the same number of simulations as MCTS then used the same best-child metric to traverse the tree to the next true state node (or the deepest intermediate state if a true state was not reached). Agent order was randomized between true states.

At each step during an experiment, each algorithm performed its respective optimization and took a single joint action. This process was repeated until a terminal state was reached or the maximum number of time steps was reached. The parameters for our experiments are shown in Table I. Except for the number of simulations, these parameters were taken directly from [13]; no hyper-parameter tuning was conducted. While [13] uses the child with the highest number of visits, N_n, rather than the highest $\bar{Q}_n$ to determine the best child, our experiments showed a greater success rate with $\bar{Q}_n$, so we used that metric.

We coded each algorithm in Python. The nodes used in MCTS and MLAT-R were represented by instances of a customized node class object. In addition to the data previously discussed, each node stored a pointer to its parent and each of its children. All child nodes were created when a node was expanded.

Our first set of experiments varied the size of the grid from 3x3 to 15x15 while keeping the number of agents constant at 3. We ran 25 experiments for each variation, and the results are shown in Fig. 4. An algorithm "failed" if the number of time steps reached four times the length of the grid without

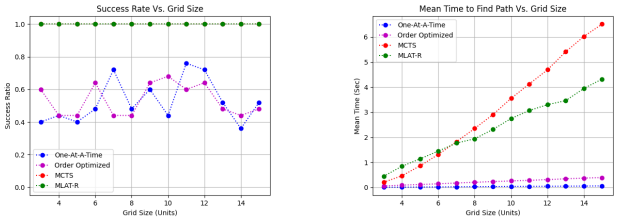

Fig. 4: Results: 3 Agents, 100 Monte Carlo Simulations

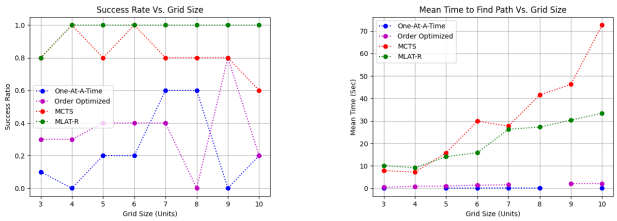

Fig. 5: Results: 4 Agents, 200 Monte Carlo Simulations

the agents reaching a terminal state. "Success rate" is the ratio of the number of experiments in which an algorithm reached a terminal state to the total number of experiments conducted. "Mean time" refers to the average time required for the algorithm to traverse the grid from the start state to a terminal state. Only the times from successful experiments are used, so long, failed runs do not skew the average.

Despite only seeing around 1/3 of the true states as MCTS (since MCTS and MLAT-R used the same number of simulations), our algorithm was able to arrive at a terminal state 100% of the time (as was MCTS). As expected, MLAT-R was more time efficient than MCTS for large grids. MCTS was slightly more time-efficient for small grids because the depth of the joint-action tree for such a small grid was very shallow. While the one-at-a-time and order-optimized algorithms were very time-efficient, they only succeeded in achieving a terminal state 53% and 54% of the time, respectively.

For our 4-agent experiments, we considered a smaller grid range: 5x5 to 10x10 as the full MCTS experiments were very time-consuming. The results are shown in Fig. 5. The one-at-a-time algorithm achieved a terminal state 24% of the time, the order-optimized algorithm achieved it 35% of the time, and MCTS achieved it 83% of the time. MLAT-R outperformed all of them, reaching a terminal state 98% of the time. We suspect MLAT-R outperformed MCTS for 4 agents because the MLAT grows deeper than the joint-action tree from MCTS for the same number of simulations due to the fewer number of actions at each level.

By 5 agents, MCTS required over an hour to move five steps, so it was no longer included in our experiments for the sake of time. On a 5x5 grid using 400 simulations, MLAT-R achieved a terminal state 60% of the time, requiring on average 252.7 seconds to do so. The one-at-a-time algorithm reached a terminal state only 4% of the time, but did so in 0.58 seconds. The order-optimized algorithm never reached a terminal state.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we offered a proof sketch of the policy improvement property of MLAT-R and compared the algorithm's performance to various state-of-the-art rollout algorithms when faced with a poor base policy. MLAT-R outperformed all three algorithms in terms of ability to reach a terminal state while considering an exponentially smaller action space than MCTS.

There are several opportunities to expand upon this work. We did not explore hyper-parameter tuning (including the number of simulations used to build the tree), branch pruning, or parallelizing the tree construction, all of which could significantly improve the algorithm's efficiency. Future work could also include using MLAT-R in approximate policy iteration. The algorithm's performance should be explored in more realistic scenarios like those considered in related works. It would also be valuable to examine MLAT-R's performance with many more agents. While we speculate MLAT-R will perform well in partially observable environments due to the Monte Carlo approach to building the MLAT, this has yet to be demonstrated. Future work could also include optimizing the algorithm's software implementation to avoid costly and unnecessary overhead (e.g. use C rather than Python, create nodes only when they need to be visited, etc.). We believe with the proper implementation, we could use this algorithm real-time in highly dynamic environments such as fleets of aerial vehicles conducting search and rescue.

ACKNOWLEDGMENT

We would like to thank our reviewers for their valuable insights and comments.

Research was sponsored by the United States Air Force Research Laboratory and the Department of the Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Department of the Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

REFERENCES

[1] J. L. Adler and V. J. Blue, "A cooperative multi-agent transportation management and route guidance system," *Transportation Research Part C: Emerging Technologies*, vol. 10, no. 5, pp. 433–454, 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X0200030X

[2] S. Wang, J. Wan, D. Zhang, D. Li, and C. Zhang, "Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination," *Computer Networks*, vol. 101, pp. 158–168, 2016, industrial Technologies and Applications for the Internet of Things. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128615005046

[3] A. Rahman, A. Bhattacharya, T. Ramachandran, S. Mukherjee, H. Sharma, T. Fujimoto, and S. Chatterjee, "Adversar: Adversarial search and rescue via multi-agent reinforcement learning," 2022.

[4] A. Henshall and S. Karaman, "Generalized multiagent reinforcement learning for coverage path planning in unknown, dynamic, and hazardous environments," in *AIAA SCITECH 2024 Forum*, 2024, p. 2762.

[5] G. Tesauro, "Temporal difference learning and td-gammon," *Commun. ACM*, vol. 38, pp. 58–68, 1995. [Online]. Available: https://api.semanticscholar.org/CorpusID:6023746

[6] G. Tesauro and G. R. Galperin, "On-line policy improvement using monte-carlo search," in *NIPS*, 1996. [Online]. Available: https://api.semanticscholar.org/CorpusID:10886094

[7] D. Bertsekas, *Network optimization: continuous and discrete models*. Athena Scientific, 1998, vol. 8.

[8] ——, "Multiagent reinforcement learning: Rollout and policy iteration," *IEEE/CAA Journal of Automatica Sinica*, vol. 8, no. 2, pp. 249–272, 2021.

[9] ——, *Lessons from AlphaZero for Optimal, Model Predictive, and Adaptive Control*. Athena Scientific, 2022.

[10] S. Bhattacharya, S. Kailas, S. Badyal, S. Gil, and D. Bertsekas, "Multiagent rollout and policy iteration for pomdp with application to multi-robot repair problems," in *Conference on Robot Learning*. PMLR, 2020, pp. 1814–1828.

[11] ——, "Multiagent reinforcement learning: Rollout and policy iteration for pomdp with application to multi-robot problems," *IEEE Transactions on Robotics*, pp. 1–20, 2023.

[12] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. P. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:515925

[13] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.aar6404

[14] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[15] D. Bertsekas and J. Tsitsiklis, *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[16] D. Garces, S. Bhattacharya, S. Gil, and D. Bertsekas, "Multiagent reinforcement learning for autonomous routing and pickup problem with adaptation to variable demand," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 3524–3531.

[17] N. Dhanaraj, S. V. Narayan, S. Nikolaidis, and S. K. Gupta, "Contingency-aware task assignment and scheduling for human-robot teams," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 5765–5771.

[18] D. Shah, Q. Xie, and Z. Xu, "Non-asymptotic analysis of monte carlo tree search," 2020.

[19] P. Perick, D. L. St-Pierre, F. Maes, and D. Ernst, "Comparison of different selection strategies in monte-carlo tree search for the game of tron," in *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, 2012, pp. 242–249.

[20] M. Lanctot, V. Lisý, and M. H. M. Winands, "Monte carlo tree search in simultaneous move games with applications to goofspiel," in *Computer Games*, T. Cazenave, M. H. Winands, and H. Iida, Eds. Cham: Springer International Publishing, 2014, pp. 28–43.

[21] O. Asik, F. B. Aydemir, and H. L. Akın, "Decoupled monte carlo tree search for cooperative multi-agent planning," *Applied Sciences*, vol. 13, no. 3, 2023. [Online]. Available: https://www.mdpi.com/2076-3417/13/3/1936

[22] L. Kocsis and C. Szepesvari, "Bandit based monte-carlo planning," in *European Conference on Machine Learning*, 2006. [Online]. Available: https://api.semanticscholar.org/CorpusID:15184765

[23] L. Kocsis, C. Szepesvari, and J. Willemson, "Improved monte-carlo search," 2006. [Online]. Available: https://api.semanticscholar.org/CorpusID:9831567

[24] R. Agrawal, "Sample mean based index policies by o(log n) regret for the multi-armed bandit problem," *Advances in Applied Probability*, vol. 27, no. 4, p. 1054–1078, 1995.

[25] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, no. 2–3, p. 235–256, may 2002. [Online]. Available: https://doi.org/10.1023/A:1013689704352

[26] J.-Y. Audibert, R. Munos, and C. Szepesvári, "Exploration–exploitation tradeoff using variance estimates in multi-armed bandits," *Theoretical Computer Science*, vol. 410, no. 19, pp. 1876–1902, 2009, algorithmic Learning Theory. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S030439750900067X

[27] M. C. Fu, "A tutorial introduction to monte carlo tree search," in *2020 Winter Simulation Conference (WSC)*, 2020, pp. 1178–1193.

[28] D. P. Bertsekas, "A distributed algorithm for the assignment problem," *Lab. for Information and Decision Systems Working Paper, MIT*, 1979.

[29] D. F. Crouse, "On implementing 2d rectangular assignment algorithms," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 52, no. 4, pp. 1679–1696, 2016.