Multi-Agent Tree Search with Dynamic Reward Shaping

Alvaro Velasquez[1], **Brett Bissey**[2], **Lior Barak**[3], **Daniel Melcer**[4],
Andre Beckus[1], **Ismail Alkhouri**[3], **George Atia**[3]

[1] Air Force Research Laboratory [2] MITRE [3] University of Central Florida [4] Northeastern University
{alvaro.velasquez.1, andre.beckus}@us.af.mil, bbissey@mitre.org, {lior.barak, ialkhouri}@knights.ucf.edu,
melcer.d@northeastern.edu, george.atia@ucf.edu

Abstract

Sparse rewards and their representation in multi-agent domains remains a challenge for the development of multi-agent planning systems. While techniques from formal methods can be adopted to represent the underlying planning objectives, their use in facilitating and accelerating learning has witnessed limited attention in multi-agent settings. Reward shaping methods that leverage such formal representations in single-agent settings are typically static in the sense that the artificial rewards remain the same throughout the entire learning process. In contrast, we investigate the use of such formal objective representations to define novel reward shaping functions that capture the learned experience of the agents. More specifically, we leverage the automaton representation of the underlying team objectives in mixed cooperative-competitive domains such that each automaton transition is assigned an expected value proportional to the frequency with which it was observed in successful trajectories of past behavior. This form of dynamic reward shaping is proposed within a multi-agent tree search architecture wherein agents can simultaneously reason about the future behavior of other agents as well as their own future behavior.

1 Introduction

In recent years, two areas of research that enable the specification of temporally extended objectives as well as the computation of decision-making policies that can similarly reason about such objectives have gained significant traction. First, the use of heuristic search planning algorithms that enable the agent to look ahead via the known agent-environment dynamics has revolutionized the area of autonomous gameplay, particularly in board games (Silver et al. 2016, 2017b,a; Schrittwieser et al. 2019). The most popular such method is Monte Carlo Tree Search (MCTS) aided by the adoption of deep Convolutional Neural Networks (CNNs) to determine the expected value of unknown states encountered during the lookahead procedure. Second, the development of formal logics over finite traces and their equivalent representation as Deterministic Finite Automata (DFA) enable the structured representation of temporally extended non-Markovian objectives. Such logics include LTL_f and LDL_f and have catalyzed various efforts on non-Markovian reinforcement learning and planning (De Giacomo and Vardi 2015; Camacho et al. 2018). In this paper, we propose a multi-agent approach that lies at the intersection of these two areas and presents a reward shaping mechanism by which a set of potentially non-Markovian objectives given as DFAs can be used to dynamically provide an artificial reward signal to each agent during the lookahead procedure of a multi-agent MCTS algorithm, henceforth referred to as Multi-Agent Tree Search (MATS). *Crucially, our approach does not simply apply MCTS to the product state space resulting from the underlying decision process and the DFA objectives.* Rather, we propose a novel dynamic reward shaping function that aids the MATS procedure to converge more quickly to higher expected values. Dynamic reward shaping was first introduced in (Velasquez et al. 2021) for single-agent MCTS. We demonstrate how cooperative and competitive behavior can arise within and across teams by sharing the same search tree in MATS as well as sharing the same DFA objective within the respective teams. The shared tree allows agents to reason about predictive optimal decisions of other agents as part of the lookahead search, whereas the shared team DFA facilitates delegation of work and cooperation for agents within the same team as some agents may be incentivized to solve different subgoals of the DFA objective as the result of prior successful experience. For example, one agent may learn to obtain a key for its team while another agent in the team learns to use this key to satisfy the team objective. We demonstrate the effectiveness of our reward shaping technique in yielding greater rewards and shorter episode lengths within MATS on randomized gridworld problems when compared to a vanilla MATS baseline which does not use this reward shaping.

In practice, our approach can be leveraged on decision processes with a high-level objective that be represented as an automaton. By leveraging the automaton representation of the objective, it is possible to speed up the convergence to good decision-making policies via the proposed automaton-guided reward shaping. On the other hand, our approach can also be applied to decision processes with a reward signal in lieu of a direct objective. Indeed, there are approaches to learning the automaton representation of non-Markovian reward signals in decision processes (Gaon and Brafman 2020). Once that automaton is learned, our reward shaping can be leveraged on it to learn what the automaton tran-

sition values are in order to provide intermediate rewards. There are also syntactic sugars which can translate a high-level natural language objective into an logic formula and its equivalent automaton representation (Brunello, Montanari, and Reynolds 2019).

It is worth noting that the proposed reward shaping approach differs from traditional reward shaping (Ng, Harada, and Russell 1999; Wiewiora, Cottrell, and Elkan 2003; Devlin and Kudenko 2012) in one key way. As opposed to changing the reward signal of the underlying decision process directly, we instead provide artificial rewards to the agents during the lookahead tree search procedure of MATS. This is done in order to exploit the lookahead properties of MATS both within the tree as well as within the agent DFA objectives. For the remainder of this paper, we do not belabor this semantic distinction.

2 Preliminaries

We assume that the agent-environment dynamics are modeled by a multi-agent Non-Markovian Reward Decision Process (NMRDP) (Thiébaux et al. 2006) with a separate reward signal for each agent. These function similarly to a Markov Decision Process (MDP), with the exception of reward signals that depend on the history of each agent.

(Multi-Agent Non-Markovian Reward Decision Process (NMRDP)) A multi-agent Non-Markovian Reward Decision Process (NMRDP) is a non-deterministic probabilistic process represented by the tuple $\mathcal{M} = (S, s_0, A, T, R, P)$, where S is a set of states, s_0 is the initial state, A is a set of actions, $T(s'|s, a) \in [0, 1]$ denotes the probability of transitioning from state s to state s' when action a is taken, and $R^p : S^* \to \mathbb{R}$ is a reward observed for a given trajectory of behavior by agent $p \in \{1, \ldots, P\}$, where P is the number of players. We denote by S^* the set of possible state sequences and $A(s) \subseteq A$ the actions available in s.

The initial state s_0 corresponds to some agent p. Once this agent chooses an action $a \in A(s_0)$, we observe a new state $s \sim T(\cdot|s_0, a)$ and it is then the turn of the next agent to choose an action. This iterative turn-taking continues cyclically over the set of agents. For readability, we represent the state and action spaces S, A as shared by all agents. Note that the definition of NMRDP closely resembles that of a Markov Decision Process (MDP). However, the non-Markovian reward formulation $R^p : S^* \to \mathbb{R}$ (often denoted by $R^p : (S \times A)^* \times S \to \mathbb{R}$ in the literature) depends on the history of agent behavior, thereby disabling the use of traditional MDP solution techniques. We encode the given non-Markovian objective using a DFA, which is a universal representation of regular languages.

[Deterministic Finite Automaton (DFA)] A DFA is a tuple $\mathcal{A} = (\Omega, \omega_0, \Sigma, \delta, F)$, where Ω is the set of nodes with initial node $\omega_0 \in \Omega$, $\Sigma = 2^{AP}$ is an alphabet defined over a given set of atomic propositions AP, $\delta : \Omega \times \Sigma \mapsto \Omega$ is the transition function, and $F \subseteq \Omega$ is the set of accepting nodes.

A relation must be made between the state space of the given NMRDP and the agent objectives represented as DFAs. This is accomplished by mapping the state space of the given NMRDP $\mathcal{M} = (S, s_0, A, T, R, P)$ to the alphabet of the corresponding DFA objective $\mathcal{A}^p = (\Omega^p, \omega_0^p, \Sigma, \delta^p, F^p)$ for each player $p \in \{1, \ldots, P\}$. More precisely, we define the labeling function $L : S \to \Sigma$. Intuitively, the label $L(s)$ of a state $s \in S$ denotes the presence or absence of certain salient features of the state space. This, in turn, may elicit a transition in $\mathcal{A}^p$ from ω to ω' if the agent is currently in some arbitrary node ω and the transition function is $\delta(\omega, L(s)) = \omega'$. This naturally allows us to define traces, or sequences of nodes in Ω^p, that correspond to trajectories, or sequences of states in S. More specifically, given a trajectory $\vec{s} = (s_0, s_{i_1}, s_{i_2}, \ldots, s_{i_n})$, we have the corresponding trace $\vec{\omega} = (\omega_0, \omega_{j_1}, \omega_{j_2}, \omega_{j_3}, \ldots, \omega_{j_{n+1}})$, where $\omega_{j_1} = \delta(\omega_0, L(s_0)), \omega_{j_2} = \delta(\omega_{j_1}, L(s_{i_1})), \omega_{j_3} = \delta(\omega_{j_2}, L(s_{i_2})), \ldots, \omega_{j_{n+1}} = \delta(\omega_{j_n}, L(s_{i_n}))$. This may include self-loops. Let $tr : S^* \to \Omega^*$ denote the mapping of a given trajectory to its corresponding trace. In the previous example, we have $tr(\vec{s}) = \vec{\omega}$. Furthermore, let last : $\Omega^* \mapsto \Omega$ denote the last node in a trace. A trace $\vec{\omega}$ is said to be accepting for player p if and only if last$(\vec{\omega}) \in F^p$. Such a trace is said to satisfy the objective encoded by the DFA $\mathcal{A}^p$ and will lead to a reward for player p for the trajectory of states that induced the trace. We now have the machinery to define the reward signals for each agent. Let $\mathcal{A}^p = (\Omega^p, \omega_0^p, \Sigma, \delta^p, F^p)$ denote the DFA objective of agent $p \in \{1, \ldots, P\}$. The reward signal $R^p : S^* \to \mathbb{R}$ for an arbitrary agent p is defined in Equation (1) for an observed trajectory $\vec{s} = (s_0, s_{i_1}, s_{i_2}, \ldots, s_{i_n})$.

$$R^p(\vec{s}) = \begin{cases} 1 & \text{last}(tr(\vec{s})) \in F^p \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

While the preceding reward formulation intuitively captures the notion of satisfying the underlying objective of an agent, it is a sparse reward signal which can make learning a useful policy difficult, particularly in temporally extended tasks. Our proposed reward shaping technique mitigates this by providing a dynamic reward signal for each transition in the agent DFAs. This reward signal captures the empirical expected value of observing a transition based on the previous experience of the agent. Empirical expected value, in this sense, denotes the proportion of time a given transition has led to a satisfying trace.

Finite-Trace Linear Temporal Logic (LTL$_f$)

Though our proposed approach is amenable to DFA objectives, it is not always straightforward to convert a high-level objective into such a representation. To that end, finite-trace Linear Temporal Logic (LTL$_f$) and Linear Dynamic Logic (LDL$_f$) have been extensively studied in recent years (De Giacomo and Vardi 2015; Camacho et al. 2018) and tools for converting formulas in these logics to DFA representations are readily available (e.g. https://pypi.org/project/flloat/). For convenience, we adopt LTL$_f$ in specifying our objectives and generating their corresponding DFAs. Given a set of atomic propositions AP, an arbitrary LTL$_f$ formula ϕ is defined inductively over the following syntactic operators, where $l \in AP$, ϕ_1, ϕ_2 are LTL$_f$ formulas, $\neg$ and $\wedge$ denote logical negation and conjunction

and **X** and **U** denote the *next* and *until* operators.

$$\phi := \text{true} \mid l \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \mathbf{X}\,\phi \mid \phi_1\,\mathbf{U}\,\phi_2$$

The core operators give rise to other useful operations, such as the *eventually* and *always* operators $\mathbf{F}\,\phi = \text{true}\,\mathbf{U}\,\phi$ and $\mathbf{G}\,\phi = \neg\mathbf{F}\,\neg\phi$, respectively. While these operators may appear limited in scope, they afford a rich description language for modeling complex behavioral specifications of systems and translating them into an equivalent DFA representation.

3 Related Work

In recent years, powerful MCTS variants using deep CNNs have been proposed for the game of Go (Silver et al. 2016) (Silver et al. 2017b), various other board games (Silver et al. 2017a) (Anthony, Tian, and Barber 2017), and Atari (Schrittwieser et al. 2019). The use of multi-agent MCTS has also been explored in (Zerbel and Yliniemi 2019), wherein each agent has its own search tree. MCTS has also been applied to learn options expressed as LTL goals (e.g. follow, wait, stop) such that, once an option is determined, deep reinforcement learning is used to learn a policy for that option (Paxton et al. 2017). These differ from our multi-agent tree search in that we reason about the mixed cooperative-competitive setting and all agents in our approach share a search tree, thereby leading to more robust adversarial and collaborative behavior as the result of predictive evaluations of other agents within the same search tree.

The use of automata to enable reward shaping in reinforcement learning and planning solutions has been explored using temporal difference learning (Sadigh et al. 2014), value iteration (Hasanbeig, Abate, and Kroening 2018), neural fitted Q-learning (Hasanbeig, Abate, and Kroening 2019), and traditional Q-learning (Hahn et al. 2019). The aforementioned efforts focus on trajectories of infinite length with objectives encoded using Linear Temporal Logic and are represented by Deterministic Rabin Automata (DRA), which require a more sophisticated acceptance condition than that of DFAs. More specifically, the reward shaping performed using a DRA typically involves an on-the-fly computation of the product transition system derived from the underlying MDP and the DRA. These approaches provide a reward for the agent upon entering a satisfying region of the product MDP, but, unlike our approach, they do not provide dynamic rewards based on the learned experience of the agent. We refer to such reward shaping as static.

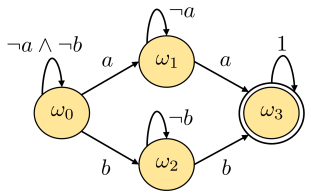

Figure 1: Simple automaton with $AP = \{a, b\}$ and $F = \omega_3$.

Static reward shaping approaches which leverage DFAs have been explored in (De Giacomo et al. 2019), (Camacho et al. 2017) by using LTL or LTL$_f$ automata to provide intermediate rewards based on the number of transitions from the current node ω to an accepting node in F in said DFA. Dynamic programming approaches over DFAs, or their more general reward machine formalism, have also been explored by treating transitions to nodes in F as rewardful and deriving expected transition and node values in the standard way using Q-learning or value iteration (Icarte et al. 2018; Camacho et al. 2019). Due to their static nature, the preceding reward shaping approaches present some limitations on how informative the reward shaping function can be since agent experience is not taken into account. Indeed, consider the simple automaton in Figure 1. Note that, from the starting node ω_0, both outgoing transitions to ω_1 and ω_2 are equally favored by the preceding approaches due to the symmetry of the automaton. However, it may be the case that in the underlying NMRDP, it is unlikely to reach a state labeled b a second time, but reaching two states labeled a is easily achievable. From the experience of the agent, such cases could be handled by using a dynamic reward shaping approach like the one presented in this paper, where the transition values in the automaton are refined based on the natural exploration of the agent as part of the planning process. As a result, the reward shaping values of the a-transitions would be much higher than those of the b-transitions. In this paper, we extend the dynamic reward shaping approach presented in (Velasquez et al. 2021) to handle multiple agents and account for both deterministic and stochastic transitions in the underlying NMRDP. It is worth noting that multi-agent reward machines have been considered recently in the context of reinforcement learning (Neary et al. 2021). We leave for future work the integration of such multi-agent reward machines with the dynamic reward shaping method proposed herein.

4 Multi-Agent Tree Search with Dynamic DFA Reward Shaping

Given an NMRDP $\mathcal{M} = (S, s_0, A, T, R, P)$ and a DFA objective $\mathcal{A}^p = (\Omega^p, \omega_0^p, \Sigma, \delta^p, F^p)$ for each agent $p \in \{1, \ldots, P\}$, our proposed reward shaping maintains statistics about how often a given transition in the automaton of each player p leads to a successful episode, thus attributing rewards to important states and actions corresponding to the trajectory $\vec{s}$ of a given trace $tr(\vec{s})$ in the DFAs. This reward shaping is integrated into a multi-agent MCTS algorithm, called MATS. We leverage the natural lookahead properties of MCTS to find future trajectories that contain automaton transitions that are more often a part of a successful episode, and bias the action selection to take the agent along this trajectory. In the sequel, we first provide a high-level explanation of the underlying processes, then proceed to formalize the individual components and present the MATS algorithm with our reward shaping function. The full algorithm can be found in the appendix.

MATS can be seen as an extension of MCTS wherein each level in the tree corresponds to the turn of exactly one agent p. Thus, MATS is equivalent to MCTS when there is only a single agent. For any given state s, MATS generates a tree of simulated experience by looking ahead using a policy π^p_{tree} for each agent p and evaluating future states. Once

enough simulated experience is gathered through the Selection, Expansion, and Update phases of the lookahead search, the agent can make an informed decision in the "real world" during the Play phase by using a different policy π_{play}. A new state is then observed in the real world and simulated experience can be collected again via the expansion of a new tree. We explain these phases in the sequel, noting that our proposed reward shaping function is applied during the Selection phase and its values are updated after every episode to reflect experience acquired by the agent in the real world following the policy π_{play}. In this sense, player p uses π^p_{tree} to plan ahead and π_{play} to take an action in the real world if it is its turn.

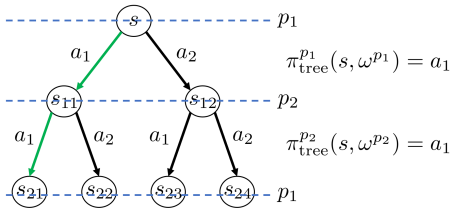

Figure 2: Example expanded tree

Selection Phase: Given a state s and the current positions $\{\omega^p\}_p$ in the DFA objectives of each agent, suppose we have already explored and evaluated various future trajectories as shown in the tree in Figure 2. The purpose of the Selection phase in MATS is to plan ahead by choosing actions, starting from the root state s until some new state (i.e. one which is not currently in the tree) is encountered. Since we have multiple agents, or players, each level in the tree corresponds to the turn of a different agent. In this example, it is the turn of player p_1 at the root of the tree. This agent makes a selection according to its tree policy $\pi^{p_1}_{\text{tree}}$ given by Equation (2). It is then the turn of player p_2, who similarly makes a selection using its tree policy $\pi^{p_2}_{\text{tree}}$. Assuming there are only two players, it is then again the turn of player p_1. The functions $Q^p(s,a)$ and $U^p(s,a)$ denote the action value and Upper-Confidence Bound (UCB) exploration functions which are defined in the Update phase. These follow the same formulation as that used in (Silver et al. 2017b). The proposed reward shaping function $R^p(s,a,\omega)$ is defined as follows:

function $R_{\mathcal{A}^p}(s,a,\omega)$:
begin
  $val1 := \sum_{s'} T(s'|s,a) Q_{\mathcal{A}^p}(\omega, \delta(\omega, L(s')))$
  $val2 := 0$
  for $s' \in \{s' | T(s'|s,a) > 0\}$ *and s' in tree* **do**
    $\omega' := \delta(\omega, L(s'))$
    $v := T(s'|s,a) \left[\max_{a' \in A(s')} R_{\mathcal{A}^p}(s', a', \omega')\right]$
    if $v > val2$ **then**
      $val2 := v$
  return $\max\{val1, val2\}$

Intuitively, this reward shaping considers the current state s and position ω in the DFA objective of agent p and evaluates the outcome of selecting action a. After taking action a, a new state s' is observed with probability $T(s'|s,a)$, thereby causing the corresponding transition from ω to ω' in the DFA objective of the agent. With each such transition, there is a dynamic value $Q_{\mathcal{A}^p}(\omega, \omega')$ associated with it which captures how often the transition has been part of a satisfying trace for that agent. However, the proposed reward shaping can reason about future transitions in the DFA as well, not just the immediate transitions. This is due to the natural lookahead properties captured by the currently expanded tree. Indeed, from the potential next state s', the reward shaping function considers possible actions and their corresponding transitions in the DFA. This process continues until the maximum DFA transition value $Q_{\mathcal{A}^p}$ is found and weighted according to the probability of reaching said transition. An illustrative example of this process can be seen in Figure 3, where it is clear that the proposed dynamic reward shaping is applied during the Selection phase of the tree search procedure. In practice, hyperparameters c_{UCB} and c_{RS} can be used to fine-tune the influence of exploration and reward shaping, respectively.

$$\pi^p_{\text{tree}}(s,\omega) = \underset{a \in A(s)}{\text{argmax}}\ Q^p(s,a) + U^p(s,a) + R_{\mathcal{A}^p}(s,a,\omega) \tag{2}$$

$$U^p(s,a) = \pi^p_{\text{CNN}}(a|s) \frac{\sqrt{\sum_{b \in A(s)} N(s,b)}}{1 + N(s,a)} \tag{3}$$

Expansion: Once a new state is encountered during the Selection phase, its value must be determined from the perspective of each agent p. This is known as the Expansion phase. While traditional MCTS approaches often leverage Monte Carlo sampling techniques to compute an empirical estimate of this value, more modern implementations use a CNN to derive a predictive value based on previous experience. In the example in Figure 2, assume that player p_1 selected action a_1, leading to state s_{11}. Then, player p_2 selected action a_1, leading to state s_{21}. Now, assume player p_1 takes action a_2 in this state and we observe an arbitrary new state s_{31}. This state is input to the CNN parameterized by θ^p for each agent p as a tensor and its predicted value $V^p_{\text{CNN}}(s_{31})$ and predicted optimal policy distribution $\pi^p_{\text{CNN}}(\cdot|s_{31})$ are obtained for each player. State s_{31} is then added to the tree. It is worth noting that, during the turn of agent p, we assume access to the CNNs of all other players as part of the training phase of the algorithm. In doing so, the MATS procedure learns effective strategies for each agent that can be validated during the deployment or testing phases, where access to the learning models of other agents is, of course, prohibited. This is a critical distinction between training and testing as during training we must make some assumption on how adversaries will behave in order to drive the learning process. This is a common assumption in multi-agent solutions that employ centralized training and decentralized execution (CTDE) (Nguyen, Nguyen, and Nahavandi 2020).

Update: The values $V^p_{\text{CNN}}(s_{31})$ and $\pi^p_{\text{CNN}}(\cdot|s_{31})$ obtained during the Expansion phase are then used to update the action value $Q^p(s,a)$ and exploration $U^p(s,a)$ functions, respectively, for each player p for the state-action pairs (s,a) leading from the root state to the newly expanded state s_{31} in the tree. In this case, these correspond

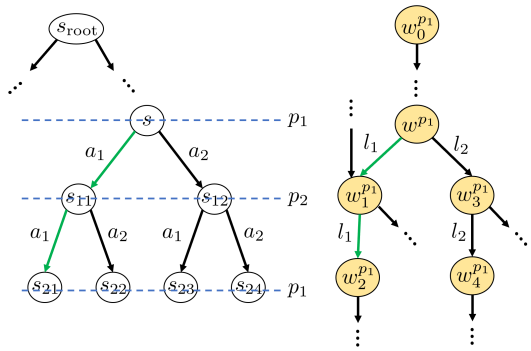

Figure 3: (*left*) Suppose the Selection phase is currently on state s of the NMRDP and node ω^{p_1} of $\mathcal{A}^{p_1}$. Let $Q_{\mathcal{A}^1}(\omega^{p_1}, \omega_1^{p_1}) = Q_{\mathcal{A}^1}(\omega^{p_1}, \omega_3^{p_1}) = Q_{\mathcal{A}^1}(\omega_3^{p_1}, \omega_4^{p_1}) = 0.05$ and $Q_{\mathcal{A}^1}(\omega_1^{p_1}, \omega_2^{p_1}) = 0.1$. Let $L(s_{11}) = L(s_{21}) = \{l_1\}$ and $L(\cdot) = \{l_2\}$ for all other states. It follows that, if the agent selects action a_1 in state s, then its DFA (*right*) will transition from ω^{p_1} to $\omega_1^{p_1}$. Recall that the value of this transition is 0.05. Alternatively, if the agent chooses action a_2, then the automaton will transition from ω^{p_1} to $\omega_3^{p_1}$, whose corresponding value is also 0.05. From this short-sighted perspective, the reward shaping function has no basis by which to favor a particular action. However, thanks to the lookahead search of MCTS, we can similarly look ahead in the automaton to transitions corresponding to later actions in the tree. For example, note that if the agent does take action a_1 in state s and action a_1 is taken again in state s_{11}, we have a transition to state s_{21}, whose label is $\{l_1\}$. This would cause transitions from ω^{p_1} to $\omega_1^{p_1}$ and then to $\omega_2^{p_1}$. The value of this latter transition is 0.1. Thus, the reward shaping $R_{\mathcal{A}^p}(s, \omega^{p_1}, a_1)$ will return $Q_{\mathcal{A}^1}(\omega_1^{p_1}, \omega_2^{p_1}) = 0.1$ and $R_{\mathcal{A}^p}(s, \omega^{p_1}, a_2)$ will return 0.05.

to $(s, a_1), (s_{11}, a_1), (s_{21}, a_2)$. Each action value function $Q^p(s, a)$ is updated to reflect the ratio $W^p(s, a)/N(s, a)$, where $N(s, a)$ denotes the number of times (s, a) has been observed during the Selection phase and $W^p(s, a)$ denotes the accumulated expansion values $V^p_{\text{CNN}}(s')$ of all states s' reached from taking action a in state s. Exploration function $U^p(s, a)$ uses the expansion output π^p_{CNN} to update exploration per Equation (3). After this Update phase, the Selection phase begins again at the root state.

Play: After a user-defined number of iterations of Selection, Expansion, and Update phases, known as the expansion limit, is reached, an action is taken in the "real world" by leveraging the simulated experience acquired during these phases. Using Figure 4 as a reference, this action follows the play policy $\pi_{\text{play}}(a|s_0) = N(s_0, a)/\sum_b N(s_0, b)$, as is done

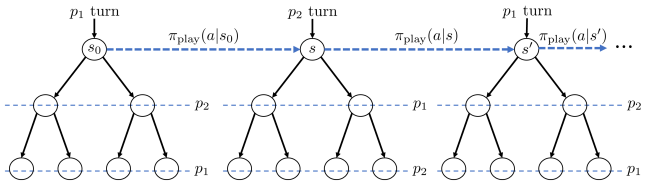

Figure 4: Example play steps using policy π_{play} in the real world by leveraging simulated experience using π^p_{tree}.

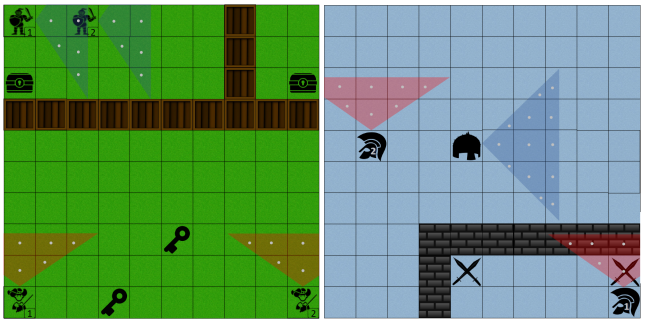

Figure 5: Instances of the *infiltrators and guards* (*left*) and *gladiators and goliath* (*right*) domains.

in (Silver et al. 2017b). A new state s is observed and a tree can be expanded anew with s at the root corresponding to the turn of player p_2. Again, the new tree is formed through the Selection, Expansion, and Update phases until the expansion limit is reached and an action in the "real world" is selected using the play policy $\pi_{\text{play}}(a|s)$. A new state s' is observed and the process continues. The episode ends once an agent p reaches its acceptance set F^p in its DFA objective.

Once an episode concludes, we use the experience obtained from the play policy in order to update the automaton transition values $Q_{\mathcal{A}^p}(\omega, \omega')$ used by $R_{\mathcal{A}^p}$ as well as train the CNN parameters θ^p of each agent p. The former are updated to reflect the ratio $W_{\mathcal{A}^p}(\omega, \omega')/N_{\mathcal{A}^p}(\omega, \omega')$ of the number of times a transition from ω to ω' was observed in a satisfying trace to the total number of times that transition was observed. Let $\vec{s}$ denote the trajectory of play steps in the episode. The CNN parameters θ^p are trained on a dataset of play steps of the form (s, a, s', r), where $r = 1$ for all play steps involved if $\text{last}(tr(\vec{s})) \in F^p$ and $r = 0$ otherwise. The formal procedure can be seen in Algorithm 1 in the appendix. Therein, the function EXPAND-TREE is called to carry out the Selection, Expansion, and Update phases.

Note that, while the proposed reward shaping function $R_{\mathcal{A}^p}$ allows for each agent to have a DFA objective, it also allows for the possibility of team objectives shared across all agents in a team. Indeed, this can be accomplished by simply having each agent in a team traverse the same DFA representation of the team objective. That is, for two agents p_1 and p_2 belonging to the same team, we have $\mathcal{A}^{p_1} = \mathcal{A}^{p_2}$. As we demonstrate in the next section, this use of a shared DFA can lead to interesting intra-team cooperative and inter-team competitive behavior. We demonstrate the utility of our approach using such team objectives by defining two multi-agent scenarios called *infiltrators and guards* and *gladiators and goliath*, where collaboration can emerge as a powerful strategy to satisfy the team DFA objective. In these environments, the agents have 6 possible actions corresponding to moves in any of the cardinal directions, an action to interact with the environment, and no-op. We consider both deterministic and stochastic transition dynamics.

5 Experimental Results

In order to evaluate the efficacy of the proposed reward shaping within MATS against that of MATS without reward

shaping, we first train two DFA teams against one another and two non-DFA teams against one another for the *infiltrators and guards* and *gladiators and goliath* domains defined below. By (non-)DFA teams, we mean teams with(out) our proposed DFA-based reward shaping function $R_{\mathcal{A}^p}$. For non-DFA teams, we set $R_{\mathcal{A}^p} = 0$ for all possible inputs.

Infiltrators and guards: This environment is defined over the atomic propositions $AP = \{k_1, k_2, c_1, c_2, sc_1, sc_2, i_1, i_2\}$ with labeling function $L(s)$ defined as follows for an arbitrary state $s \in S$: $L(s) = k_j$ holds iff infiltrator j holds a key in state s, c_j holds iff infiltrator j obtained a treasure chest, sc_j holds iff treasure chest j is in the cone of visibility of one of the guards, and i_j holds iff infiltrator j is in the cone of visibility of one the guards. This environment consists of a team of two infiltrators and a team of two guards. Each agent has a cone of visibility as seen in Figure 5. The goal of the infiltrator team is to first obtain one of two keys in the map by standing on a key tile. Afterwards, one of the infiltrators must interact with one of the two treasures on the map in order to unlock it with the key that has been obtained. This objective is given by the LTL$_f$ formula $\mathbf{F}(k_1 \vee k_2) \wedge (\mathbf{F}(k_1 \vee k_2)\ \mathbf{U}\ \mathbf{F}(c_1 \vee c_2))$. The interact action also allows infiltrators to permanently immobilize a guard when standing directly behind them or to place a box from their inventory of two boxes each on their current position in order to block the visibility of guards. Guards can make use of the interact action to destroy all adjacent boxes, but only when both guards are on the same tile. The objective of the guard team is to foil the infiltrators by finding both infiltrators and monitoring the treasure to ensure it has not been stolen. This objective is given by the LTL$_f$ formula $(\mathbf{G}(\mathbf{F}(sc_1)) \wedge\ \mathbf{G}(\mathbf{F}(sc_2))) \wedge \mathbf{F}(i_1) \wedge \mathbf{F}(i_2)$. An example of the Selection, Expansion, and Update phases for this environment can be seen in Figure 9 at the end of this paper.

Gladiators and goliath: This environment is defined over the atomic propositions $AP = \{gw_1, gw_2, gG_1, gG_2, Gg_1, Gg_2\}$ denoting the following: gw_j holds iff gladiator j is holding a weapon, gG_j holds iff the goliath is in the cone of visibility of gladiator j, and Gg_j holds iff gladiator j is in the cone of visibility of the Goliath. This environment consists of a team of two gladiators facing off against one goliath whose cone of visibility is greater than that of the gladiators. The objective of the gladiators is to each obtain a weapon by standing on a weapon tile and both gladiators must have the goliath in their sights simultaneously at some point. This objective is given by the LTL$_f$ formula $\mathbf{F}(gw_1) \wedge \mathbf{F}(gw_2) \wedge \mathbf{F}(gG_1 \wedge gG_2)$ with a corresponding DFA containing 8 nodes and 27 edges. The interact action allows gladiators to erect a wall from their inventory on the tile they are standing on in order to block the visibility of the goliath. Gladiators begin with no walls in their inventory, but can pick up wall tiles by standing on them. The goal of the goliath is simply to have each gladiator in its cone of visibility for three consecutive turns. This is given by the LTL$_f$ formula $\mathbf{F}(\mathbf{X}(Gg_1 \wedge \mathbf{X}(Gg_1 \wedge \mathbf{X}(Gg_1)))) \wedge \mathbf{F}(\mathbf{X}(Gg_2 \wedge \mathbf{X}(Gg_2 \wedge \mathbf{X}(Gg_2))))$ whose corresponding DFA has 16 nodes and 49 edges.

We begin with homogeneous opponent games that run for one million play steps each. We then swap opponents, so that each DFA team is now playing against a non-DFA team, and these runs train for another million play steps. The reward curves and average episode lengths for each team can be observed in Figure 6 for both deterministic and stochastic environment dynamics. For the stochastic dynamics, if an agent chooses to move right (left), it will do so with 90% probability, but may instead move to the tile above or below it with 5% probability each. Similarly, if an agent chooses to move up (down), it will do so with 90% probability, but may instead move to the tile right or left of it with 5% probability. The deterministic dynamics are defined in the obvious sense, with $T(s'|s, a) \in \{0, 1\}$.

Through the evaluation of total reward curves, we see our DFA teams performing significantly better than their non-DFA counterparts in both domains. We also observe a degradation in performance for non-DFA teams after swapping to face a DFA opponent, demonstrating that DFA teams can learn more robust policies. After the 2M play steps for each team (1M against a DFA opponent and 1M against a non-DFA opponent) in the deterministic environment, we observe that DFA infiltrators (guards) achieved 69.49% (61.94%) higher rewards than their non-DFA counterparts. In the *gladiators and goliath* deterministic environment, we observe similar gains, with the DFA gladiators (goliath) scoring 63.73% (13.34%) higher than their non-DFA counterparts. The difference in performance is even more pronounced in the stochastic setting, where we observe that DFA infiltrators (guards) achieve 71.80% (154.07%) higher rewards than their non-DFA counterparts. In the *gladiators and goliath* stochastic environment, the DFA gladiators (goliath) score 41.11% (23.98%) higher than non-DFA counterparts. See appendix for neural network architecture details.

The proposed reward shaping also led to more data-efficient policies. Indeed, consider the average episode lengths in the homogeneous games (i.e. DFA against DFA teams, non-DFA against non-DFA teams). After the execution of the first 1M play steps, we observe that, in the *infiltrators and guards* environment, the deterministic (stochastic) DFA games have an average episode length that is 80.33% (42.64%) less than that of non-DFA counterparts. See Figure 6 for results. In the *gladiators and goliath* environment, we similarly observe deterministic (stochastic) DFA games have an average episode length that is 16.28% (34.53%) less than that of their non-DFA counterparts. [1]

Behavioral Case Studies

We examine episodes of complex behavior carried out by agents in the *infiltrators and guards* environment after training for one million play steps using our proposed reward shaping function. We first show an example of collaborative behavior executed by members of the infiltrator team. A diagram of interesting trajectories and actions taken by the learned agents is shown in Figure 7.

Initially, both infiltrators move in the direction of the keys. However, while Infiltrator 2 proceeds to collect one of the keys, Infiltrator 1 moves back to the right and then moves

[1] Code and parameters at https://github.com/bb912/MATS-DRS

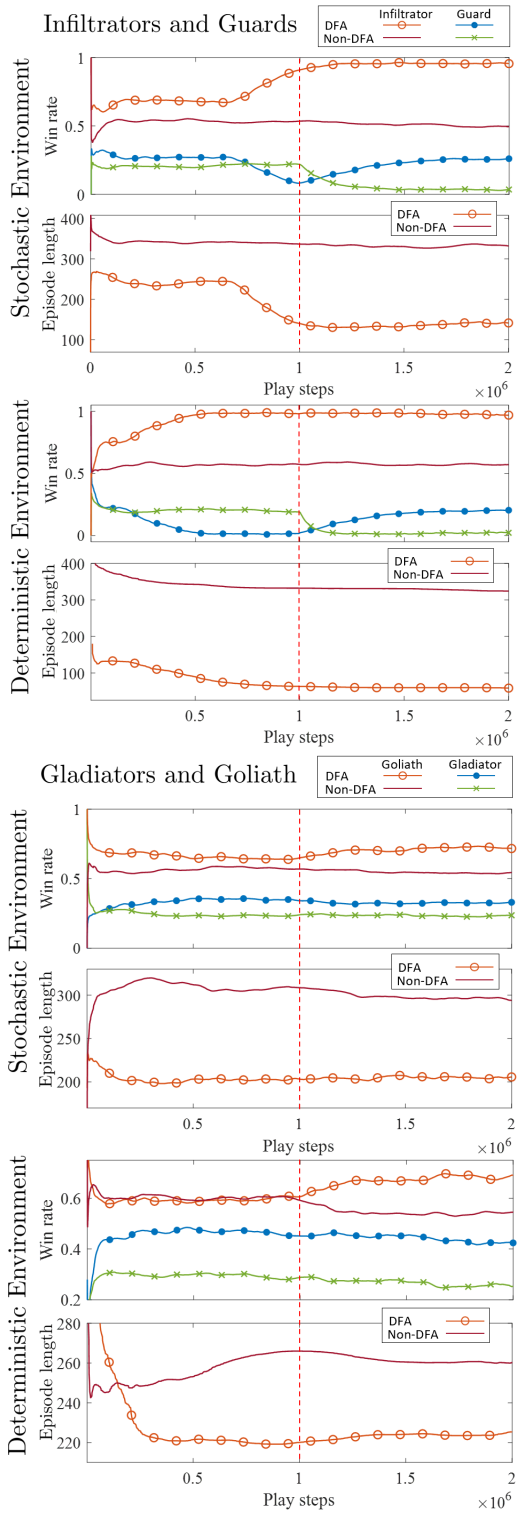

Figure 6: Performance of DFA and non-DFA teams for the (*top*) *infiltrators and guards* and (*bottom*) *gladiators and goliath* environments with stochastic and deterministic environment dynamics.

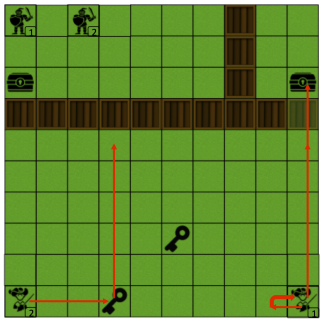

Figure 7: Example of collaborative behavior. Red arrows denote the action trajectory of our infiltrator team.

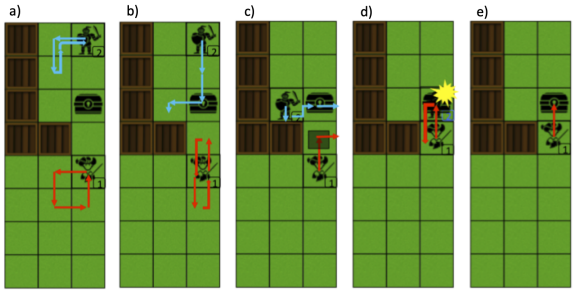

Figure 8: Example of adversarial behavior. Red arrows denote the action trajectory of Infiltrator 1 and blue arrows denote the same for Guard 2. The translucent black square in (c) denotes a No-op action. Letters $a - e$ denote consecutive sub-trajectories within a single episode. The final positions in a are the initial positions in b, and so on.

directly upward towards the closest treasure chest. We typically observe Infiltrator 1 following a similar pattern: delaying their upward motion until Infiltrator 2 gets the key, and sometimes initially moving towards the key as in this case. Because Infiltrator 2 has already obtained the key, Infiltrator 1 wins the game by reaching the treasure chest. The episode completes in 33 total play steps: 9 steps for Infiltrator 1, and 8 steps for each of the other three agents. Note that this case can also be viewed as an act of delegation, since Infiltrator 1 takes on the task of reaching the treasure chest, while Infiltrator 2 adopts the alternative task of obtaining the key.

An example of learned adversarial behavior on the part of the infiltrators can be seen in Figure 8, which displays five consecutive sub-trajectories ($a - e$). Note that these images are focused on the top-right corner of the environment. The Infiltrator team already has a key and must reach the treasure chest to win the game. Infiltrator 2 and Guard 1 remain unseen through the duration of the trajectory. If the Guard team sees both infiltrators, and then the chest, the guards will win. Notice Infiltrator 1 stalling around the chest, seemingly to avoid the guard's cone of visibility which extends 2 spaces in front of Guard 2. Guard 2 moves right towards the chest as Infiltrator 1 moves up and chooses a No-op action (c). Infiltrator 1 makes a move to kill Guard 2 (d) once the guard reaches the chest and is facing right, where the cone of visibility no longer presents a threat to the infiltrator. The infiltrator then makes a move downwards, and then back up to the chest, thereby winning the game (e).

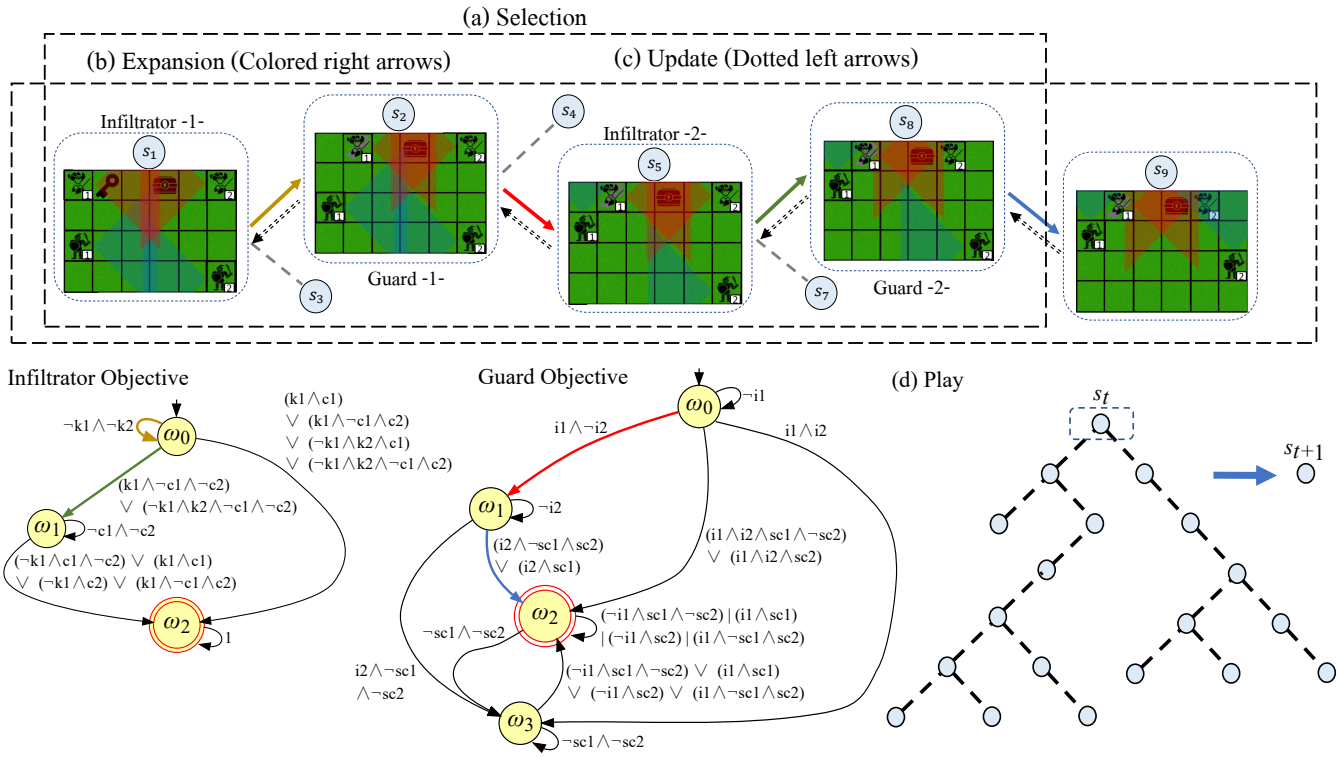

Figure 9: (*top*) MATS iteration of the Selection, Expansion, and Update phases for a 4×6 multi-agent game of *infiltrators and guards* given by the NMRDP $M = (S, A = \{\leftarrow, \downarrow, \rightarrow, \uparrow, \text{interact, no-op}\}, T, R, P = \{I, G\})$. (*bottom*) DFAs $\mathcal{A}^I = (\Omega^I, \omega_0^I, \Sigma = 2^{\text{AP}}, \delta^I, F^I)$ and $\mathcal{A}^G = (\Omega^G, \omega_0^G, \Sigma = 2^{\text{AP}}, \delta^G, F^G)$ used to model the infiltrator and guard team objectives, where transitions are visualized as Boolean formulas over AP (e.g. $\neg k1$ denotes that Infiltrator 1 is not holding a key). Each team shares a DFA as well as a CNN, where a binary matrix is used as input to each CNN in order to determine whose agent's turn it is within the team. (a) This iteration of MATS begins in the root state s_1 with the turn of Infiltrator 1. Suppose the infiltrators (yellow and green arrows in the tree and the Infiltrator DFA) and one guard (red arrow in the tree and the Guard DFA) have already made their moves according to their tree policy given by Equation (2). It is currently the turn of Guard 2, who must select which action to choose according to its tree policy. Once Guard 2 chooses its action to move up (blue arrow), this causes a transition into state s_9, where the guard successfully sees Infiltrator 2 and the treasure (i.e. $L(s_9) = \{sc_1, i_2\}$), thereby causing a transition from ω_1 to ω_2 in the guard team objective automaton. Since this corresponds to an accepting trace for the guard team, the values $V^I_{\text{CNN}}(s_9)$ and $V^G_{\text{CNN}}(s_9)$ output by the infiltrator and guard team CNNs during the expansion phase (b) are expected to be low and high, respectively. This expansion is carried out because the newly observed state s_9 is a new leaf node in the tree and we must thus determine its expected value from the point of view of all teams (including infiltrators). This is done by representing s_9 as a series of binary matrices denoting the presence of the various items in AP (e.g. the treasure chests, keys) and the cones of visibility of the agents and feeding these as inputs to a CNN for each team. This CNN outputs the expected win rates $V^I_{\text{CNN}}(s_9), V^G_{\text{CNN}}(s_9)$ of the newly expanded state from the perspective of each team and it also outputs the predicted optimal policy $\pi^I_{\text{CNN}}(\cdot|s_9), \pi^G_{\text{CNN}}(\cdot|s_9)$ (used to update the exploration function per Equation (3)) for whichever agent's turn it is. (c) The edges (s, a) in the tree consisting of the trajectory from the root s_1 to the expanded node s_9 are updated with new action values $Q^I(s, a), Q^G(s, a)$ and exploration values $U^I(s, a), U^G(s, a)$ for each team based on the expanded values and we begin a new trajectory starting at the root of the tree until some new leaf is expanded. (d) After some given threshold on the number of expansions is reached for the current MATS iteration rooted at s_1, Infiltrator 1 at the root of the tree utilizes the play policy $\pi^I_{\text{play}}(a|s_1) = N(s_1, a)/\sum_b N(s_1, b)$ to take an action in the real world. A new state s' is observed corresponding to the turn of Guard 1 and a new MATS iteration begins with root state s'.

6 Conclusion

A novel dynamic reward shaping function which leverages the Deterministic Finite Automaton (DFA) representations of multi-agent objectives was proposed and integrated within a multi-agent Monte-Carlo Tree Search implementation, called Multi-Agent Tree Search (MATS). The proposed reward shaping function leverages the lookahead properties of MATS, which simultaneously allows the agent to reason about and evaluate future states of the environment and of the DFA objective, and is dynamic in the sense that it captures the experience of the agents and is updated to reflect the empirical expected value of individual DFA transitions. We demonstrated the effectiveness of our approach on random cooperative-competitive gridworld environments by comparing against a MATS baseline with no reward shaping. The results show that our reward shaping function achieves higher rewards and shorter episode lengths than the MATS baseline.

Appendix: Algorithm Pseudocode

function EXPAND-TREE
begin
  Input: Root state s_{root}, current DFA positions $\{\omega^p\}_p$, current player $\hat{p}$
  $W^p, N^p, Q^p, U^p := 0$
  for $k := 1$ *to* $expansionLimit$ **do**
    $s := s_{\text{root}}$
    $\vec{S} := \emptyset$ `// stores trajectory`
    while s *in currently expanded tree* **do**
      $a \sim \pi_{\text{tree}}(s, \omega^{\hat{p}})$ `// selection; this is where the reward shaping function is used`
      $\vec{S} := \vec{S} \bigcup \{(s, a)\}$
      $N(s, a) := N(s, a) + 1$
      $s' \sim T(\cdot|s, a)$
      Change $\hat{p}$ to reflect the next player
      for *each player* p **do**
        $\omega^p := \delta(\omega^p, L(s'))$
      $s := s'$
    $s_{\text{expand}} := s$
    for *each player* p **do**
      $(V^p_{\text{CNN}}, \pi^p_{\text{CNN}}) := f_{\theta^p}(s_{\text{expand}})$ `// expansion to determine the value from the perspective of each player`
      for *each* (s, a) *in* $\vec{S}$ **do**
        $W^p(s, a) := W^p(s, a) + V^p_{\text{CNN}}$
        $Q^p(s, a) := W^p(s, a)/N(s, a)$ `// update tree`
        $U^p(s, a) \propto \pi^p_{\text{CNN}}/N(s, a)$ (See Equation (3))
  Return $\pi_{\text{play}}(a|s) = N(s, a)/\sum_b N(s, b)$

Acknowledgments

This research was supported in part by the Air Force Research Laboratory through the Information Directorate's Information Institute® Contract Number FA8750-20-3-1003 and FA8750-20-3-1004, the Air Force Office of Scientific Research through Award 20RICOR012, and the National Science Foundation through CAREER Award CCF-1552497 and Award CCF-2106339.

Algorithm 1: Multi-Agent Tree Search (MATS)
begin
  Input: NMRDP $\mathcal{M}$, DFAs $\mathcal{A}^p$, parameters θ^p. starting player $\hat{p}$
  Initialize $W_{\mathcal{A}^p}, N_{\mathcal{A}^p}, Q_{\mathcal{A}^p} := 0$ for each player p
  Initialize memory $\mathbb{M}^p_{\text{play}} := \emptyset, terminal_p := 0$ for each player p
  $s := s_0$ `// set initial state`
  for *each player* p **do**
    $\omega^p := \delta^p(\omega^p_0, L(s_0))$ `// DFA transitions`
  for *each episode* **do**
    $\pi_{\text{play}}(\cdot|s) :=$ EXPAND-TREE$(s, \{\omega^p\}_p, \hat{p})$
    $a \sim \pi_{\text{play}}(\cdot|s)$
    $s' \sim T(\cdot|s, a)$
    $\mathbb{M}^p_{\text{play}} := \mathbb{M}^p_{\text{play}} \bigcup \{(s, a, r = 0, s')\}$
    Change $\hat{p}$ to reflect the next player
    for *each player* p **do**
      $\mathbb{M}_{\mathcal{A}^p} := \mathbb{M}_{\mathcal{A}^p} \bigcup \{(\omega^p, \delta^p(\omega^p, L(s')))\}$
      $\omega^p := \delta^p(\omega^p, L(s'))$
      if $\omega_p \in F^p$ **then**
        for *each* $(s, a, r, s') \in \mathbb{M}^p_{play}$ **do**
          $(s, a, r, s') := (s, a, 1, s')$
          $terminal_p := 1$ `// satisfying trace for agent` p
    if $terminal_p = 1$ *for any* p **then**
      for *each player* p **do**
        Let $(\pi_{\text{CNN}}, V_{\text{CNN}}) := f_{\theta^p}(\cdot)$
        Train θ^p using loss $\mathbb{E}_{(s,a,r,s') \sim \mathbb{M}^p_{\text{play}}} \left[(r - V_{\text{CNN}})^2 - a^T \log \pi_{\text{CNN}}\right]$
        for *each* $(\omega, \omega') \in \mathbb{M}_{\mathcal{A}^p}$ **do**
          $W_{\mathcal{A}^p}(\omega, \omega') := W_{\mathcal{A}^p}(\omega, \omega') + terminal_p$
          $N_{\mathcal{A}^p}(\omega, \omega') := N_{\mathcal{A}^p}(\omega, \omega') + 1$
          $Q_{\mathcal{A}^p}(\omega, \omega') := W_{\mathcal{A}^p}(\omega, \omega')/N_{\mathcal{A}^p}(\omega, \omega')$ `// update automaton stats`
        Begin next episode with $s := s_0$ and $\omega^p := \delta^p(\omega^p_0, L(s_0))$ for each player p
      else
        go to Line 17
  return θ^p and $Q_{\mathcal{A}^p}$ for each player p

References

Anthony, T.; Tian, Z.; and Barber, D. 2017. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems*, 5360–5370.

Brunello, A.; Montanari, A.; and Reynolds, M. 2019. Synthesis of LTL formulas from natural language texts: State of the art and research directions. In *26th International Symposium on Temporal Representation and Reasoning (TIME 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Camacho, A.; Baier, J. A.; Muise, C.; and McIlraith, S. A. 2018. Finite LTL synthesis as planning. In *Twenty-Eighth International Conference on Automated Planning and Scheduling*.

Camacho, A.; Chen, O.; Sanner, S.; and McIlraith, S. A. 2017. Non-Markovian rewards expressed in LTL: guiding search via reward shaping. In *Tenth Annual Symposium on Combinatorial Search*.

Camacho, A.; Icarte, R. T.; Klassen, T. Q.; Valenzano, R. A.; and McIlraith, S. A. 2019. LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning. In *IJCAI*, volume 19, 6065–6073.

De Giacomo, G.; Iocchi, L.; Favorito, M.; and Patrizi, F. 2019. Foundations for restraining bolts: Reinforcement learning with LTLf/LDLf restraining specifications. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 128–136.

De Giacomo, G.; and Vardi, M. 2015. Synthesis for LTL and LDL on finite traces. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Devlin, S. M.; and Kudenko, D. 2012. Dynamic potential-based reward shaping. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, 433–440. IFAAMAS.

Gaon, M.; and Brafman, R. 2020. Reinforcement Learning with Non-Markovian Rewards. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 3980–3987.

Hahn, E. M.; Perez, M.; Schewe, S.; Somenzi, F.; Trivedi, A.; and Wojtczak, D. 2019. Omega-regular objectives in model-free reinforcement learning. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 395–412. Springer.

Hasanbeig, M.; Abate, A.; and Kroening, D. 2018. Logically-constrained reinforcement learning. *arXiv preprint arXiv:1801.08099*.

Hasanbeig, M.; Abate, A.; and Kroening, D. 2019. Certified reinforcement learning with logic guidance. *arXiv preprint arXiv:1902.00778*.

Icarte, R. T.; Klassen, T.; Valenzano, R.; and McIlraith, S. 2018. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*, 2107–2116.

Neary, C.; Xu, Z.; Wu, B.; and Topcu, U. 2021. Reward Machines for Cooperative Multi-Agent Reinforcement Learning. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, 934–942.

Ng, A. Y.; Harada, D.; and Russell, S. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, 278–287.

Nguyen, T. T.; Nguyen, N. D.; and Nahavandi, S. 2020. Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications. *IEEE transactions on cybernetics*, 50(9): 3826–3839.

Paxton, C.; Raman, V.; Hager, G. D.; and Kobilarov, M. 2017. Combining neural networks and tree search for task and motion planning in challenging environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 6059–6066. IEEE.

Sadigh, D.; Kim, E. S.; Coogan, S.; Sastry, S. S.; and Seshia, S. A. 2014. A learning based approach to control synthesis of Markov decision processes for linear temporal logic specifications. In *53rd IEEE Conference on Decision and Control*, 1091–1096. IEEE.

Schrittwieser, J.; Antonoglou, I.; Hubert, T.; Simonyan, K.; Sifre, L.; Schmitt, S.; Guez, A.; Lockhart, E.; Hassabis, D.; Graepel, T.; et al. 2019. Mastering Atari, Go, Chess and Shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587): 484.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. 2017a. Mastering Chess and Shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017b. Mastering the game of Go without human knowledge. *Nature*, 550(7676): 354.

Thiébaux, S.; Gretton, C.; Slaney, J.; Price, D.; and Kabanza, F. 2006. Decision-theoretic planning with non-Markovian rewards. *Journal of Artificial Intelligence Research*, 25: 17–74.

Velasquez, A.; Bissey, B.; Barak, L.; Beckus, A.; Alkhouri, I.; Melcer, D.; and Atia, G. 2021. Dynamic Automaton-Guided Reward Shaping for Monte Carlo Tree Search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 12015–12023.

Wiewiora, E.; Cottrell, G. W.; and Elkan, C. 2003. Principled methods for advising reinforcement learning agents. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 792–799.

Zerbel, N.; and Yliniemi, L. 2019. Multiagent Monte Carlo Tree Search. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, 2309–2311. International Foundation for Autonomous Agents and Multiagent Systems.