Dynamic Automaton-Guided Reward Shaping for Monte Carlo Tree Search

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

[1]Information Directorate, Air Force Research Laboratory
[2]Department of Electrical and Computer Engineering, University of Central Florida
[3]Department of Computer Science, Northeastern University
{alvaro.velasquez.1, andre.beckus}@us.af.mil, {brettbissey, lior.barak, ialkhouri}@knights.ucf.edu,
melcer.d@northeastern.edu, george.atia@ucf.edu

Abstract

Reinforcement learning and planning have been revolutionized in recent years, due in part to the mass adoption of deep convolutional neural networks and the resurgence of powerful methods to refine decision-making policies. However, the problem of sparse reward signals and their representation remains pervasive in many domains. While various reward-shaping mechanisms and imitation learning approaches have been proposed to mitigate this problem, the use of human-aided artificial rewards introduces human error, sub-optimal behavior, and a greater propensity for reward hacking. In this paper, we mitigate this by representing objectives as automata in order to define novel reward shaping functions over this structured representation. In doing so, we address the sparse rewards problem within a novel implementation of Monte Carlo Tree Search (MCTS) by proposing a reward shaping function which is updated dynamically to capture statistics on the utility of each automaton transition as it pertains to satisfying the goal of the agent. We further demonstrate that such automaton-guided reward shaping can be utilized to facilitate transfer learning between different environments when the objective is the same.

Introduction

In reinforcement learning and planning settings, a reward signal over state-action pairs is used to reinforce and deter positive and negative action outcomes, respectively. However, from a practical perspective, it is common to have a high-level objective which can be captured as a language of what would constitute good behaviors. Such objectives can be represented via various types of automata. For example, the deterministic finite automata we consider in this paper can represent any regular language and thus afford a large space of possibilities for defining objectives. If these objectives are expressed in natural language, there are translation mechanisms for obtaining the corresponding automaton (Brunello, Montanari, and Reynolds 2019). On the other hand, there are also approaches for learning the graph structure of the automaton that represents a given reward signal (Xu et al. 2020; Gaon and Brafman 2020; Toro Icarte et al. 2019), so such approaches could be used to learn the automaton structure from a reward signal.

Given an automaton representation of the underlying agent objective, we investigate how to guide and accelerate the derivation of decision-making policies by leveraging this structured representation. This use of automata allows us to reason about non-Markovian reward signals that account for the prior state history of the agent, which is useful and sometimes necessary in domains with sparse rewards or partial observability (Toro Icarte et al. 2019). In particular, we make two main contributions. First, we introduce Automaton-Guided Reward Shaping (AGRS) for Monte Carlo Tree Search (MCTS) algorithms as a means to mitigate the sparse rewards problem by leveraging the empirical expected value of transitions within the automaton representation of the underlying objective. In this sense, expected value denotes the frequency with which transitions in the automaton have been observed as part of a trajectory which satisfies the given objective. We integrate the proposed AGRS within modern implementations of MCTS that utilize deep reinforcement learning to train Convolutional Neural Networks (CNNs) for policy prediction and value estimation. Our second contribution entails the use of the aforementioned automata as a means to transfer learned information between two problem instances that share the same objective. In particular, we record the empirical expected value of transitions in the objective automaton in a simple environment and use these to bootstrap the learning process in a similar, more complex environment. We demonstrate the effectiveness of AGRS within MCTS, henceforth referred to as MCTS$_\mathcal{A}$, by comparing the win-rate curves to those of a vanilla MCTS baseline on randomized grid-world problems.

The foregoing is useful as a complement to existing approaches by reasoning over both the learned representation of agent-environment dynamics through deep CNNs as well as the learned representation of the underlying objective via automata. We argue that this is particularly useful due to the typically low dimensionality of the automaton that represents the objective. This means that individual transitions within the automaton correspond to potentially many different transitions within the environment and, hence, within the Monte Carlo tree. Thus, reward shaping defined over the former can be used to accelerate learning over many instances of the latter.

Preliminaries

We assume that the agent-environment dynamics are modeled by a Non-Markovian Reward Decision Process (NMRDP), defined below.

Definition 1 (Non-Markovian Reward Decision Process) *A Non-Markovian Reward Decision Process (NMRDP) is a non-deterministic probabilistic process represented by the tuple $\mathcal{M} = (S, s_0, A, T, R)$, 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 $R : S^* \to \mathbb{R}$ is a reward observed for a given trajectory of states. We denote by S^* the set of possible state sequences and $A(s) \subseteq A$ the actions available in s.*

Note that the definition of NMRDP closely resembles that of a Markov Decision Process (MDP). However, the non-Markovian reward formulation $R : S^* \to \mathbb{R}$ (often denoted by $R : (S \times A)^* \times S \to \mathbb{R}$ in the literature) depends on the history of agent behavior. Given an NMRDP, we define a labeling function $L : S \to 2^{AP}$ that maps a state in the NMRDP to a set of atomic propositions in AP which hold for that given state. We illustrate this with a simple example in Figure 1 (*left*). The atomic propositions AP assigned to the NMRDP can correspond to salient features of the state space and will be used to define the transition dynamics of the objective automaton as defined below.

Definition 2 (Automaton) *An automaton 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, $\delta : \Omega \times \Sigma \to \Omega$ is the transition function, and $F \subseteq \Omega$ is the set of accepting nodes. For simplicity, we use $\omega \xrightarrow{\sigma} \omega' \in \delta$ to denote that $\sigma \in \Sigma$ causes a transition from node ω to ω'.*

Given an automaton $\mathcal{A} = (\Omega, \omega_0, \Sigma, \delta, F)$, a trace is defined as a sequence of nodes $\omega_0 \xrightarrow{\sigma_{i_1}} \omega_{i_1} \xrightarrow{\sigma_{i_2}} \omega_{i_2} \xrightarrow{\sigma_{i_3}} \ldots$ starting in the initial node ω_0 such that, for each transition, we have $(\omega_{i_k}, \sigma_{i_k}, \omega_{i_{k+1}}) \in \delta$. An accepting trace is one which ends in some node belonging to the accepting set F. Such a trace is said to satisfy the objective being represented by the automaton. For the remainder of this paper, we use the terms nodes and states to refer to vertices in automata and NMRDPs, respectively.

Until now, we have used the notion of an atomic proposition set AP in defining the labeling function of an NMRDP and the alphabet of an automaton. In this paper, our focus is on deriving a policy for a given NMRDP such that it is conducive to satisfying a given objective by leveraging the automaton representation of the same, where both the NMRDP and the automaton share the same set of atomic propositions AP. As a result, an arbitrary transition from state s to s' in the NMRDP can cause a transition in the automaton objective via the observance of atomic propositions that hold in s' as determined by the NMRDP labeling function L. To illustrate the relationship between trajectories, or sequences of states, in an NMRDP and their corresponding traces, or sequences of nodes, in the automaton, recall the example in Figure 1. Note that the policy

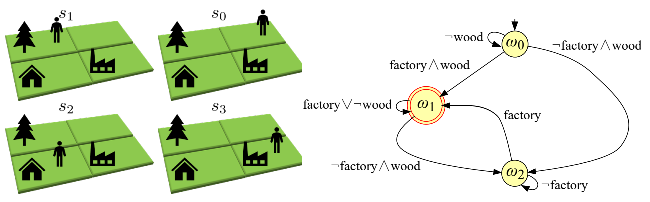

Figure 1: (*left*) NMRDP consisting of four states s_0, s_1, s_2, s_3 denoting which tile the agent is on, starting with s_0 in the top-right corner and progressing counter-clockwise. There are four actions a_1, a_2, a_3, a_4 corresponding to LEFT, DOWN, RIGHT, UP. The transition function is deterministic and defined in the obvious way (e.g. $T(s_1|s_0, a_1) = 1$). Given a set of atomic propositions $AP = \{\text{wood}, \text{factory}, \text{house}\}$, we have the following state labels: $L(s_0) = \{\}, L(s_1) = \{\text{wood}\}, L(s_2) = \{\text{house}\}, L(s_3) = \{\text{factory}\}$. That is, the labels correspond to the atomic propositions that are true in a given state. (*right*) Automaton $\mathcal{A} = (\Omega = \{\omega_0, \omega_1, \omega_2\}, \omega_0, \Sigma = 2^{\{\text{wood, factory, house}\}}, \delta = \{\omega_0 \xrightarrow{\neg\text{wood}} \omega_0, \ldots\}, F = \{\omega_1\})$ corresponding to the objective that the agent will eventually be on a tile containing wood and that, if the agent stands on said tile, then it must eventually reach a tile containing a factory. Transitions visualized with a Boolean formula are used as a shorthand notation for the sets of atomic propositions that cause that transition. For example, the transition $\omega_0 \xrightarrow{\neg f \wedge w} \omega_2$ (f for factory and w for wood) in the automaton is shorthand for transitions $(\omega_0, \sigma, \omega_2) \in \delta$ such that factory $\notin \sigma$ and wood $\in \sigma$.

$\pi(s_0) = a_1$ (Go LEFT), $\pi(s_1) = a_2$ (Go DOWN), $\pi(s_2) = a_3$ (Go RIGHT), $\pi(s_3) = \cdot$ (no-op) yields a finite trajectory of four states (s_0, s_1, s_2, s_3) whose corresponding trace in the automaton is $\omega_0 \xrightarrow{L(s_0)} \omega_0 \xrightarrow{L(s_1)} \omega_2 \xrightarrow{L(s_2)} \omega_2 \xrightarrow{L(s_3)} \omega_1$, where $L(s_0) = \{\}, L(s_1) = \{\text{wood}\}, L(s_2) = \{\text{house}\}, L(s_3) = \{\text{factory}\}$. Since $\omega_1 \in F$, this policy leads to a trajectory in the NMRDP which satisfies the objective. In order to reinforce such behavior and formalize the foregoing, we utilize a binary reward signal for the NMRDP denoting whether the underlying automaton objective has been satisfied by an observed trajectory $\vec{s} = (s_0, \ldots, s_t, s_{t+1})$. Let $tr : S^* \to \Omega^*$ and last $: \Omega^* \mapsto \Omega$ denote the mapping of a given trajectory to its corresponding trace and the last node in a trace, respectively. A trace $\vec{\omega}$ is accepting if and only if $\text{last}(\vec{\omega}) \in F$. The reward signal $R : S^* \to \{0, 1\}$ is defined in Equation (1) below.

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

Note that the use of such a reward signal makes the problem of finding optimal policies susceptible to sparse rewards. Indeed, one can envision many instances where such a reward is only accessible after a long time horizon, which can hinder learning. The proposed AGRS will mitigate this problem by providing artificial intermediate rewards that reflect the frequency with which transitions in the automaton objective have been conducive to satisfying the objective.

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

A classic problem in formal methods is that of converting a high-level behavioral specification, or objective, into a form that is amenable to mathematical reasoning. Indeed, the foregoing discussion on utilizing automata representations of objectives subsumes the idea that we have some means of performing this translation from objective to automaton. There are various ways to mitigate this problem by enforcing that the given objective be specified in some formal logic which can then be easily converted into an equivalent automaton. In order to represent the objective of an agent as an automaton, we choose the formal logic known as finite-trace Linear Temporal Logic (LTL$_f$) for its simple syntax and semantics. This logic is known to have an equivalent automaton representation (De Giacomo, De Masellis, and Montali 2014; De Giacomo and Vardi 2015; Camacho et al. 2018a). Other choices include finite-trace Linear Dynamic Logic (LDL$_f$) and regular expressions.

Given a set of atomic propositions AP, a formula in LTL$_f$ is constructed inductively using the operations $p, \neg\phi, \phi_1 \vee \phi_2, \mathbf{X}\phi, \mathbf{F}\phi, \mathbf{G}\phi, \phi_1\mathbf{U}\phi_2$, where $p \in AP$ and ϕ, ϕ_1, ϕ_2 are LTL$_f$ formulas. The operators $\neg, \vee$ denote logical negation and disjunction and **X, F, G, U** denote the *Next, Eventually, Always*, and *Until* operators, respectively, such that $\mathbf{X}\phi$ (resp. $\mathbf{F}\phi, \mathbf{G}\phi, \phi_1\mathbf{U}\phi$) holds iff ϕ holds in the next time step (resp. some time in the future, all times in the future, at some point and ϕ_1 is true until that point). An example LTL$_f$ formula is $(\mathbf{F}\text{ wood}) \wedge (\mathbf{F}\text{ wood} \implies \mathbf{F}\text{ factory})$, which corresponds to the automaton objective of Figure 1 (*right*).

For the remainder of this paper, we will use the automaton $\mathcal{A}$ of a given LTL$_f$ objective to define a reward-shaping function reflecting the frequency with which individual transitions in $\mathcal{A}$ were part of some accepting trace. In particular, we integrate this reward-shaping function within MCTS in order to leverage the power of modern MCTS solvers and exploit their lookahead property to simulaneously look ahead in the NMRDP and the automaton objective.

Related Work

Given an infinite-trace LTL objective, reinforcement learning has been applied to learn controllers 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). These approaches reward the agent for reaching an accepting component of the automaton, but they do not yield intermediate rewards to mitigate the sparse rewards problem.

More related to our approach is the use of reward machines (Icarte et al. 2018) as described in Definition 3. These correspond to automata-based representations of the reward function. By treating these reward machines as a form of abstract NMRDP, reward shaping functions have been defined in the literature (Icarte et al. 2018; Camacho et al. 2019). In particular, note that the nodes and transitions of such automata can be treated as states and actions in Q-learning and value iteration approaches. Such methods have been proposed and yield a value for each transition in the automaton. Furthermore, optimal policy invariance (Ng, Harada, and Russell 1999) can be established for tabular settings.

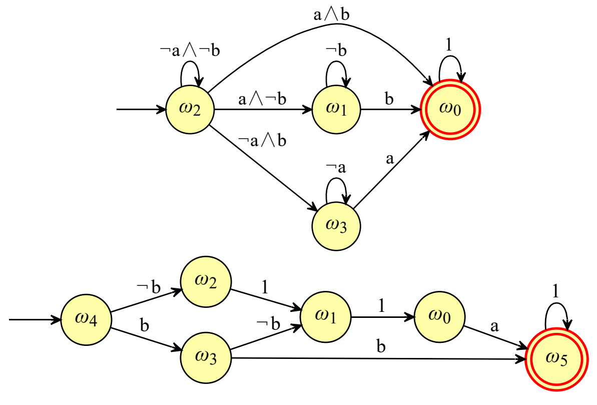

Figure 2: Simple example automata.

Definition 3 (Reward Machine) *A reward machine is an automaton given by the tuple $\mathcal{A} = (\Omega, \omega_0, \Sigma, \delta, \mathcal{R}, \rho)$, 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, $\delta : \Omega \times \Sigma \to \Omega$ is the transition function, $\mathcal{R}$ is an output alphabet denoting the possible reward values, and $\rho : \Omega \times \Sigma \to \mathcal{R}$ is the output function.*

Consider the reward shaping function obtained from performing value iteration over a reward machine representation of the objective given by the formula $\phi = (\mathbf{F}a) \wedge (\mathbf{F}b)$, whose automaton is given in Figure 2 (*top*). The reward machine has $\mathcal{R} = \{0, 1\}$ with a reward value of 1 being observed if ϕ is satisfied. This yields the output transition values of $\rho(\omega_2, a \wedge b) = \rho(\omega_1, b) = \rho(\omega_3, a) = 1$ and $\rho = 0$ for all other transitions. Consider value iteration over the function below as proposed in (Camacho et al. 2019), where initial values are set to 0.

$$V(\omega) := \max_{\omega'=\delta(\omega,\sigma)} \rho(\omega, \sigma) + \gamma V(\omega') \tag{2}$$

Due to the symmetry of traces $\omega_2 \to \omega_1 \to \omega_0$ and $\omega_2 \to \omega_3 \to \omega_0$, the same values would be obtained for each of those corresponding transitions, which may be an uninformative reward shaping mechanism in some settings. Indeed, one can envision arbitrary environments where observing a followed by b may be much more efficient than observing b followed by a. However, this type of reward shaping would not exploit such knowledge.

Another approach using automata for reward shaping was proposed in (Camacho et al. 2017, 2018b), where instead of solving a planning problem over a reward machine, reward shaping is defined over each automaton transition to be inversely proportional to the distance between a given node and an accepting node in the automaton of an LTL objective. Like the previous example, this introduces some uninformative and potentially unproductive rewards. Indeed, consider the LTL objective given by $\phi = (\mathbf{XXX}a) \vee (b \wedge \mathbf{X}b)$ whose automaton is shown in Figure 2 (*bottom*).

Note that the aforementioned reward shaping will favor transitions $\omega_4 \xrightarrow{b} \omega_3, \omega_3 \xrightarrow{b} \omega_5$ in the automaton over transitions $\omega_4 \xrightarrow{\neg b} \omega_2, \omega_3 \xrightarrow{\neg b} \omega_1$ since the distance to accepting node ω_5 is greater for the latter. However, again we can envision scenarios where rewards defined in this way are uninformative. For example, consider an NMRDP where a node labeled a is three steps away from the initial state, but the second observance of b is many steps away.

In contrast, we focus on a dynamic reward shaping function that changes based on observed experience in that each transition value in the automaton captures the empirical expected value associated with said transition. This is based on the experience collected by the agent, thereby implicitly reflecting knowledge about the given environment in a way that the more static reward shaping mechanisms previously mentioned cannot. MCTS lends itself well for the adoption of such an approach given that each search tree is the culmination of many trajectories in the NMRDP and its corresponding traces in the automaton objective. Thus, tree search provides a natural way to refine the estimated values of each automaton transition via simulated experience before the agent makes a real decision in the environment. In particular, the lookahead property of MCTS naturally allows the same within a given objective automaton. We exploit this lookahead in the proposed AGRS function and apply it within a modern MCTS implementation using deep CNNs similar to those proposed for the game of Go (Silver et al. 2016) (Silver et al. 2017), various other board games (Silver et al. 2018) (Anthony, Tian, and Barber 2017), and Atari (Schrittwieser et al. 2020).

We also explore the use of our proposed AGRS as a transfer learning mechanism. Despite the tremendous success of transfer learning in fields like natural language processing (Devlin et al. 2018) and computer vision (Weiss, Khoshgoftaar, and Wang 2016), transfer learning in the fields of reinforcement learning and planning faces additional challenges due in part to the temporally extended objectives of the agents and the fact that actions taken by these agents can affect the probability distribution of future events. The area of curriculum learning (Narvekar et al. 2020) addresses this by decomposing the learning task into a graph of sub-tasks and bootstrapping the learning process over these first. For example, this may include learning chess by first ignoring all pawns and other pieces, and progressively adding more complexity as the agent learns good policies. State and action mappings or embeddings have also been explored for handling transfer between different environments (Taylor and Stone 2009; Chen et al. 2019). These can also be used to derive similarity metrics between the state spaces of different environments. In this paper, we avoid the standard modus operandi of focusing on state spaces or neural network weights to enable transfer learning and instead focus on commonality found in the underlying *goals* of the agent as given by their automaton representation.

Monte Carlo Tree Search with Automaton-Guided Reward Shaping

For the purposes of the MCTS$_\mathcal{A}$ algorithm, we are interested in finding a policy for a given NMRDP $\mathcal{M} = (S, s_0, A, T, R)$ which satisfies a given LTL$_f$ specification ϕ whose automaton is defined by $\mathcal{A} = (\Omega, \omega_0, \Sigma, \delta, F)$. As such, we must establish a connection between the agent-environment dynamics defined by $\mathcal{M}$ and the environment-objective dynamics defined by $\mathcal{A}$. Let $\omega \xrightarrow{L(s)} \omega'$ denote that the observance of labels, or features, $L(s)$ (e.g., "safe", "hostile", etc.) of state s (e.g., an image) in any trajectory of $\mathcal{M}$ causes a transition from ω to ω' in $\mathcal{A}$ (see Figure 1 for an example). This establishes a connection between the NMRDP $\mathcal{M}$ and the automaton $\mathcal{A}$. In particular, note that any trajectory of states $\vec{S} = (s_0, \ldots, s_t)$ in $\mathcal{M}$ has a corresponding trace $\vec{\Omega} = (\omega_0, \ldots, \omega_t)$ in $\mathcal{A}$.

MCTS functions by (a) simulating experience from a given state s_t by looking ahead in $\mathcal{M}$ and (b) selecting an action to take in s_t based on predicted action values from the simulated experience (see Algorithm 1). For (a), a search tree is expanded by selecting actions according to the tree policy (3) starting from the root s_t until a leaf in the tree is encountered (Lines 7 through 12). The tree policy consists of an action value function Q and an exploration function U as formulated in modern MCTS implementations (Silver et al. 2017), where c_{UCB} is a constant used to control the influence of exploration. The Y function in (3) corresponds to the proposed AGRS function discussed later in this section. Once an action is taken in a leaf state (Line 13), a new state is observed (Line 14) in the form of a newly added leaf s^{expand}. The value $V_{\text{CNN}}(s^{\text{expand}})$ of this new leaf as well as its predicted action probabilities $\pi_{\text{CNN}}(a|s^{\text{expand}})$ are given by the CNN of the agent (Line 15). This is known as the Expansion phase of MCTS. After expansion, the Update phase begins, wherein the values of the edges (s, a) corresponding to actions on the path leading from the root s_t to the new leaf node s^{expand} are updated (Lines 18 through 21). These values are $\{N(s,a), W(s,a), Q(s,a), \pi_{\text{CNN}}(a|s)\}$, where $N(s,a)$ denotes the number of times action a has been taken in state s in the search tree, $W(s,a)$ is the total cumulative value derived from $V_{\text{CNN}}(s')$ for all leafs s' that were eventually expanded when action a was taken in state s, and $Q(s,a) = W(s,a)/N(s,a)$ is the action value function yielding the empirical expected values of actions in a given state. After this Update phase, a new iteration begins at the root of the tree. The tree expansion and update process repeats until a user-defined limit on the number of expansions is reached (Line 2). After this process of simulating experience (a) is done, (b) is achieved by taking an action in the real world at the root s_t of the tree according to the play policy $\pi_{\text{play}}(a|s_t) \propto N(s_t, a)/\sum_b N(s_t, b)$ (Line 22) as used in (Silver et al. 2016) due to lowered outlier sensitivity when compared to a play policy that maximizes action value (Enzenberger et al. 2010). The foregoing corresponds to a single call of the lookahead process given by Algorithm 1. Algorithm 2 utilizes multiple lookahead calls in sequence in order to solve the given NMRDP by taking actions in sequence using the play policy returned by Algorithm 1 and using the data generated from these transitions to update the CNN of the agent. Indeed, using Algorithm 2 as a reference, note that once an action is taken using the play policy (Line 9), a new state s_{t+1} is observed (Line 10) and the process begins again with s_{t+1} at the root of a new tree. Samples of the form $(s_t, \pi_{\text{play}}(\cdot|s_t), r, s_{t+1})$ are derived from such actions following the play policy π_{play} and are stored (Line 14) in order to train the CNN (Line 21), where $r = R((s_0, \ldots, s_{t+1}), \pi_{\text{play}}(\cdot|s_t))$ denotes the reward observed. As defined in equation (1), we have $r = 1$ iff the trajectory corresponding to the sequence of states from s_0 to s_{t+1} yields a satisfying trace in the given automaton objective (see Definition 2) and $r = 0$ otherwise. The entire process can be repeated over many episodes (Line 3) until the performance of π_{play} converges.

$$\pi_{\text{tree}}(s, \omega) = \text{argmax}_{a \in A} \left(Q(s, a) + U(s, a) + Y(s, a, \omega)\right) \quad (3)$$

$$U(s, a) = c_{\text{UCB}} \pi_{\text{CNN}}(a|s) \frac{\sqrt{\sum_{b \in A(s)} N(s, b)}}{1 + N(s, a)} \quad (4)$$

$$Y(s, a, \omega) = c_{\mathcal{A}} \max \left(V_{\mathcal{A}}(\omega, \omega'), \max_{a' \in A(s')} Y(s', a', \omega') \right) \quad (5)$$

For the AGRS component, we define a reward shaping function $Y : S \times A \times \Omega \to [0, 1]$ which is used as part of the tree policy in equation (3). The AGRS function $Y(s, a, \omega)$ given by equation (5) recursively finds the maximum automaton transition value anywhere in the subtree rooted at state s, where ω is the corresponding automaton node that the agent is currently in at state s and we have $T(s'|s, a) > 0, (\omega, L(s'), \omega') \in \delta$. This automaton transition value is given by $V_{\mathcal{A}}(\omega, \omega') = W_{\mathcal{A}}(\omega, \omega')/N_{\mathcal{A}}(\omega, \omega')$, where $W_{\mathcal{A}}(\omega, \omega')$ denotes the number of times a transition $\omega \to \omega'$ in the automaton has led to an accepting trace in the automaton and $N_{\mathcal{A}}(\omega, \omega')$ denotes the total number of times that transition has been observed. The constant $c_{\mathcal{A}}$ is used to control the influence of AGRS on the tree policy. Note that these values are updated after every episode in Algorithm 2 (Lines 16 through 19).

We compare the foregoing proposed MCTS$_{\mathcal{A}}$ approach against a modern vanilla MCTS approach using two grid-world environments *Blind Craftsman* and *Treasure Pit* defined in the next section and visualized in Figure 3. In these environments, an agent has 6 possible actions corresponding to moves in any of the cardinal directions, an action to interact with the environment, and no-op.

Experimental Results

We evaluate MCTS$_{\mathcal{A}}$ using 10×10 and 25×25 grid-world environments defined in the sequel. For each instance of a 10×10 environment, the object layout of the grid-world is randomly generated and remains the same for every episode. We compute the average performance and variance of 100 such instances using MCTS$_{\mathcal{A}}$ and a vanilla MCTS baseline that does not use the AGRS function Y (i.e., $Y(s, a, \omega) = 0$, for all s, a, ω). Each instance is trained for 30,000 play steps, corresponding to a varying number of episodes per instance. The 25×25 environments will be used in Section 6 to demonstrate the effectiveness of the AGRS function as a transfer learning mechanism from 10×10 to 25×25 instances that share the same objective automaton.

Algorithm 1: Lookahead$_{\mathcal{A}}$

```
begin
       Input: NMRDP $\mathcal{M}$, state $s$, Automaton $\mathcal{A}$, node
         $\omega$, NN $f_\theta$
 1     $W(s,a), N(s,a), Q(s,a) := 0$
 2     for $k := 1$ to expansionLimit do
 3         $t := 0$
 4         $s_t := s$
 5         $X_{\mathcal{M}} := \emptyset$   // stores transitions
 6         $\omega_t := \delta(\omega, L(s_t))$
 7         while $s_t$ not a leaf node do
 8             $a \sim \pi_{\text{tree}}(s_t, \omega_t)$   // selection
 9             $s_{t+1} \sim T(\cdot|s_t, a)$
10             $\omega_{t+1} := \delta(\omega_t, L(s_{t+1}))$
11             $X_{\mathcal{M}} := X_{\mathcal{M}} \bigcup \{(s_t, a, s_{t+1})\}$
12             $t := t + 1$
13         $a \sim \pi_{\text{tree}}(s_t, \omega_t)$   // selection
14         $s^{\text{expand}} \sim T(\cdot|s_t, a)$
15         $(V_{\text{CNN}}(s^{\text{expand}}), \pi_{\text{CNN}}(\cdot|s^{\text{expand}})) :=$
             $f_\theta(s^{\text{expand}})$   // expansion
16         $X_{\mathcal{M}} := X_{\mathcal{M}} \bigcup \{(s_t, a, s^{\text{expand}})\}$
18         for Each $(s, a, s')$ in $X_{\mathcal{M}}$ do
19             $W(s,a) := W(s,a) + V(s^{\text{expand}})$
20             $N(s,a) := N(s,a) + 1$
21             $Q(s,a) := W(s,a)/N(s,a)$
                // update tree
22     Return $\pi_{\text{play}}(a|s)$
```

Algorithm 2: Monte-Carlo Tree Search with Automaton-Guided Reward Shaping MCTS$_{\mathcal{A}}$

```
begin
       Input: NMRDP $\mathcal{M}$, Automaton $\mathcal{A}$, NN $f_\theta$
 1     $W_{\mathcal{A}}(\omega, \omega'), N_{\mathcal{A}}(\omega, \omega'), V_{\mathcal{A}}(\omega, \omega') := 0$
 2     $Mem := \emptyset$
 3     for each episode do
 4         $t := 0$
 5         $s_t := s_0$
 6         $\omega_t := \delta(\omega_0, L(s_0))$
 7         $X_{\mathcal{A}} = \emptyset$   // stores transitions
 8         while $s_t$ is not terminal do
 9             $a \sim$ Lookahead$_{\mathcal{A}}(\mathcal{M}, s_t, \mathcal{A}, \omega_t, f_\theta)$
10             $s_{t+1} \sim T(\cdot|s_t, a)$
11             $\omega_{t+1} := \delta(\omega_t, L(s_{t+1}))$
12             $r \sim R(s_t, a, s_{t+1})$
13             $X_{\mathcal{A}} := X_{\mathcal{A}} \bigcup \{(\omega_t, \omega_{t+1})\}$
14             $Mem := Mem \bigcup \{(s_t, a, r, s_{t+1})\}$
15             $t := t + 1$
16         for Each $(\omega, \omega')$ in $X_{\mathcal{A}}$ do
17             $W_{\mathcal{A}}(\omega, \omega') :=$
                 $W_{\mathcal{A}}(\omega, \omega') + R(s_{t-1}, a, s_t)$
18             $N_{\mathcal{A}}(\omega, \omega') := N_{\mathcal{A}}(\omega, \omega') + 1$
19             $V_{\mathcal{A}}(\omega, \omega') := W_{\mathcal{A}}(\omega, \omega')/N_{\mathcal{A}}(\omega, \omega')$
                // update automaton stats
20         Let $(p, v) := f_\theta(s)$
21         Train $\theta$ using loss
             $\mathbb{E}_{(s,a,r,s') \sim Mem} \left[(r - v)^2 - a^T \log p\right]$
```

The CNN used during the expansion phase of MCTS$_\mathcal{A}$ to obtain $\pi_{\text{CNN}}, V_{\text{CNN}}$ consists of a shared trunk with separate policy and value heads. The trunk contains four layers. The first two are convolutional layers with a 5×5 kernel, 32 and 64 channels, respectively, and ELU activation. The next two layers are fully connected with ReLU activation; the first is of size 256 and the second of size 128. The value and policy head each contain a fully connected layer of size 128 with ReLU activation. The value head then contains a fully connected layer of size 1 with sigmoid activation, while the policy head contains a fully connected layer of size 6, with softmax. The input to the neural network consists of a number of stacked 2D layers that are the size of the board. One layer contains a one-hot representation of the position of the agent. There is a layer for every tile type (i.e., wood, factory, etc.) with a value of 1 in the positions where that tile is placed in the environment and 0 otherwise. Lastly, there is a layer for each inventory item type (i.e., wood, tool) with a value between 0 and 1 as determined by the current number of held item type divided by the maximum capacity of that item. Maximum capacities differ between environments.

Environments

Blind Craftsman: This environment is defined over the atomic propositions $AP = \{\text{wood, home, factory, tools} \geq 3\}$. The agent must first collect wood, then bring it to the factory to make a tool from the wood. After the agent has crafted at least three tools and arrived at the home space, it has satisfied the objective. The CNN of the agent is trained on the full grid-world tensor, however the automaton is only shaped by the labels of the spaces the agent stands on. If the agent is standing on a wood tile and chooses the interact action, the wood inventory is increased by one and the tile disappears. If the agent is on top of a factory tile and interacts, the wood inventory decreases by one and the finished product, or tool, inventory increases by one. We restrict the agent to hold a maximum of two woods at a time, making it so that the agent must go back and forth between collecting wood and visiting the factory. The objective is given by the LTL$_f$ formula $\mathbf{G}(\text{wood} \implies \mathbf{F}\ \text{factory}) \wedge \mathbf{F}(\text{tools} \geq 3\ \wedge\ \text{home})$ with a corresponding automaton of 3 nodes and 9 edges. See Figure 4 for results. This minecraft-like environment is similar to that proposed in (Toro Icarte et al. 2018).

Treasure Pit: This environment is defined over the atomic propositions $AP = \{\text{a, b, c, pit}\}$. The goal of the agent in this environment is to collect the treasures a, c, and then b. The treasure a will be closest to the agent. Treasure c will be farthest, and there will be a pit in between the two containing treasure b. Once the agent enters the pit, it cannot leave. Hence, the objective requires the agent to enter the pit already holding a and c, and stay in the pit until it picks up b to win. The environment generation algorithm limits the size of the pit area to allow at least one non-pit space on each side of the board. This ensures that it will not be impossible for our agent to maneuver around the pit and collect a and c before entering the pit. The objective is given by the LTL$_f$ formula $\mathbf{F}\ \text{a} \wedge \mathbf{F}\ \text{c} \wedge \mathbf{F}\ \text{b} \wedge \mathbf{F}(\ \mathbf{G}\ \text{pit})$. It has a corresponding automaton of 5 nodes and 17 edges. See Figure 5 for results.

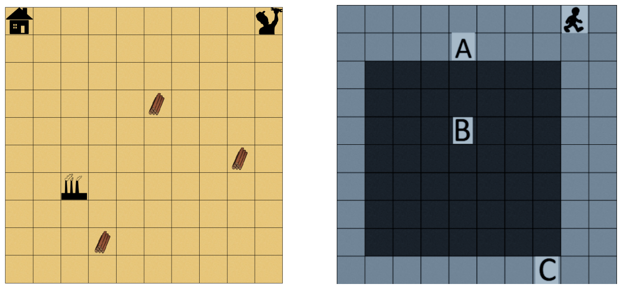

Figure 3: Possible 10×10 instances of *Blind Craftsman* (*Left*) and *Treasure Pit* (*Right*).

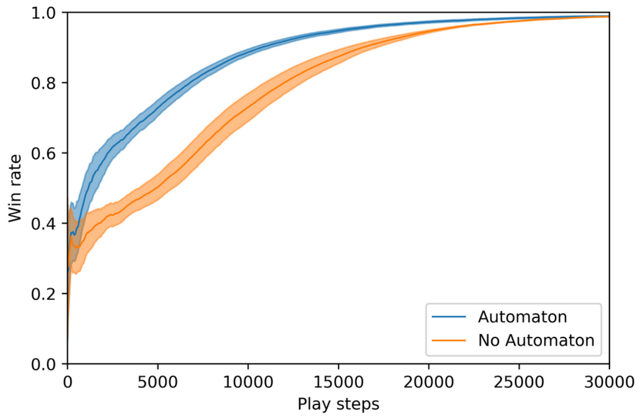

Figure 4: *Blind Craftsman* average performance of MCTS$_\mathcal{A}$ against a vanilla MCTS baseline. Average win rate and variance are reported for 100 fixed instances of the environment. Win rate refers to the rate at which play steps were part of a trajectory that satisfied the LTL$_f$ objective.

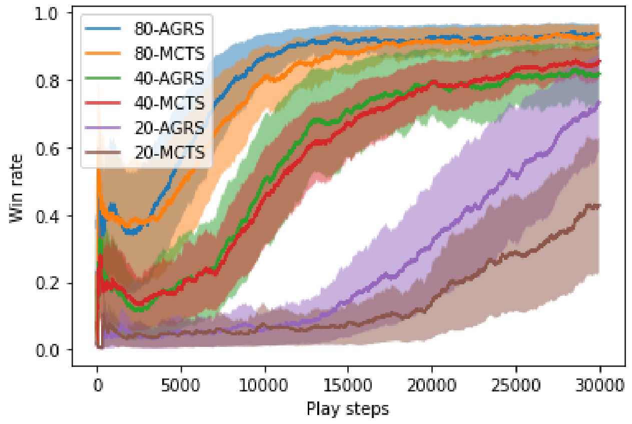

Figure 5: Win rate comparison for MCTS leaf expansion limits of 20, 40, and 80 expansions per call to Algorithm 1 for agents completing the *Treasure Pit* Objective. At 80 and 40 MCTS expansions, MCTS$_\mathcal{A}$ yields a slightly steeper win rate curve when compared to vanilla MCTS before reaching convergence around 15,000 play steps. At 20 expansions, MCTS$_\mathcal{A}$ greatly outperforms vanilla MCTS. In practice, the optimal expansion limit will depend on computing resources and objective complexity.

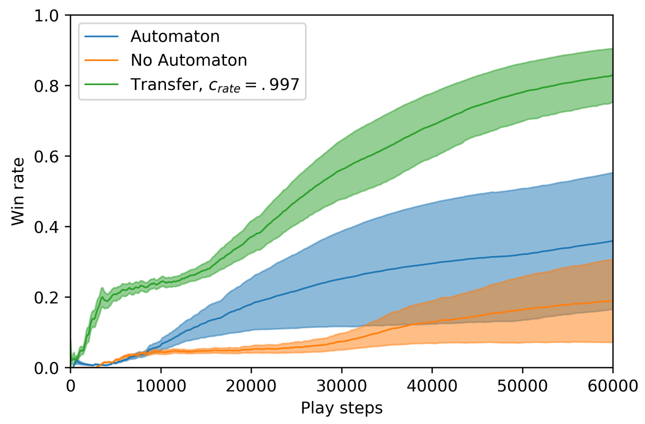

Figure 6: The performance of MCTS$_\mathcal{A}$ with transfer learning versus MCTS$_\mathcal{A}$ without transfer learning and vanilla MCTS (No Automaton) as a function of play steps executed in the 25×25 *blind craftsman* environment. Average win rate and variance are reported for 25 randomly generated instances of the environment.

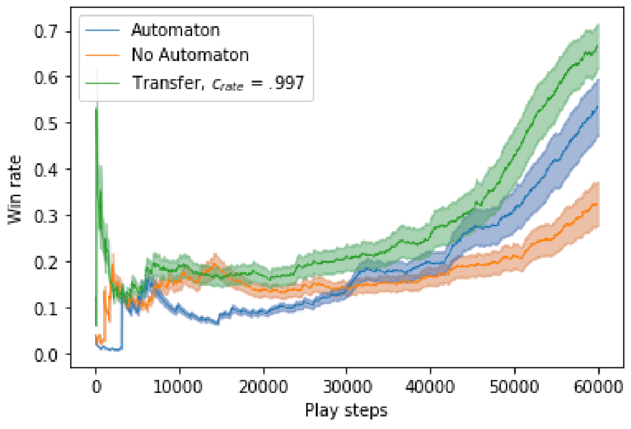

Figure 7: The win rate and variance are reported for MCTS$_\mathcal{A}$ with transfer learning, MCTS$_\mathcal{A}$ without transfer learning, and vanilla MCTS (No Automaton) as a function of the number of play steps executed in 25 instances of 25×25 *Treasure Pit* environments.

Transfer Learning

We also investigated using the automaton transition values $V_\mathcal{A}$ of the objective automaton as a transfer learning mechanism between different environment instances that share the same objective. A more complex version of the *Blind Craftsman* and *Treasure Pit* environments is generated by scaling the grid size to 25×25. Given a 25×25 instance of an environment, we attempt to transfer learned transition values from a simpler 10×10 version of the environment. The automaton-guided transfer learning procedure works as follows. First, the automaton transition values $V_\mathcal{A}^{10\times10}$ of the objective automaton $\mathcal{A}$ are computed (Line 19 in Algorithm 2) for many randomly generated episodes of the 10×10 environment corresponding to a total of 500K play steps. The exposure to these random grid layouts for a given environment helps to compute more robust expected automaton transition values. When a 25×25 instance is started, in addition to initializing its automaton values $V_\mathcal{A}^{25\times25}$ to 0, we load the pre-computed automaton values $V_\mathcal{A}^{10\times10}$. The latter are used to accelerate the learning process in the 25×25 environment by exploiting the prevalence of successful transitions as seen in simpler 10×10 environments that share the same objective.

It is worth noting that statistics are still collected for the automaton corresponding to the 25×25 environment with value function $V_\mathcal{A}^{25\times25}$ that was initialized at the beginning of the 25×25 instance. However, when MCTS$_\mathcal{A}$ queries the automaton transition values while computing the AGRS function $Y(\cdot)$ during the node selection phase (Line 13 in Algorithm 1), we anneal between the 10×10 values $V_\mathcal{A}^{10\times10}$ and the new 25×25 values $V_\mathcal{A}^{25\times25}$ according to the current episode number, t_{ep}, as seen in equation (6). This transfer process allows for simple generalizations about an environment's relationship with the objective structure to be learned on a smaller scale first. The automaton statistics learned from the 10×10 grid guide the early learning process for all 25×25 environment runs.

The benefits of the automaton-guided transfer learning is evident in the *Blind Craftsman* and *Treasure Pit* environments, as demonstrated in Figures 6 and 7.

$$V_\mathcal{A}(\cdot) = \eta_{t_{\text{ep}}} V_\mathcal{A}^{10\times10}(\cdot) + (1-\eta_{t_{\text{ep}}}) V_\mathcal{A}^{25\times25}(\cdot), \eta_{t_{\text{ep}}} = (c_{\text{rate}})^{t_{\text{ep}}} \tag{6}$$

We train the 25×25 environment for 60k play steps, using the same hyperparameters and network architecture as our 10×10 environments. This training process is repeated 25 times and the average win rates and variances of (i) utilizing MCTS$_\mathcal{A}$ with transfer learning, (ii) utilizing MCTS$_\mathcal{A}$ without transfer learning, and (iii) utilizing vanilla MCTS are reported in Figures 6 and 7. In both of these instances, we observe the agent leveraging the transferred automaton statistics has a higher and steeper win rate curve than MCTS$_\mathcal{A}$, and MCTS$_\mathcal{A}$ performs better on average than vanilla MCTS. We hypothesize that automaton-guided transfer learning has the most advantage when facilitating the scheduling of subgoals over complex board structures. We train the 25x25 environment with the same hyper-parameters and network architecture as in the previous section.

Conclusion

MCTS$_\mathcal{A}$ is proposed as a means to integrate novel reward shaping functions based on automaton representations of the underlying objective within modern implementations of Monte Carlo Tree Search. In doing so, MCTS$_\mathcal{A}$ simultaneously reasons about both the state representation of the environment and the automaton representation of the objective. We demonstrate the effectiveness of our approach on structured grid-world environments, such as those of *Blind Craftsman* and *Treasure Pit*. The automaton transition values computed using MCTS$_\mathcal{A}$ are shown to be a useful transfer learning mechanism between environments that share the same objective but differ in size and layout.

Acknowledgements

This project is supported in part by the US Army/DURIP program W911NF-17-1-0208 and NSF CAREER Award CCF-1552497.

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. 2018a. 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.; Chen, O.; Sanner, S.; and McIlraith, S. A. 2018b. Non-Markovian rewards expressed in LTL: Guiding search via reward shaping (extended version). In *GoalsRL, a workshop collocated with ICML/IJCAI/AAMAS*.

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.

Chen, Y.; Chen, Y.; Yang, Y.; Li, Y.; Yin, J.; and Fan, C. 2019. Learning action-transferable policy with action embedding. *arXiv preprint arXiv:1909.02291* .

De Giacomo, G.; De Masellis, R.; and Montali, M. 2014. Reasoning on LTL on finite traces: Insensitivity to infiniteness. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*.

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, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* .

Enzenberger, M.; Müller, M.; Arneson, B.; and Segal, R. 2010. Fuego—an open-source framework for board games and Go engine based on Monte Carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games* 2(4): 259–270.

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.

Narvekar, S.; Peng, B.; Leonetti, M.; Sinapov, J.; Taylor, M. E.; and Stone, P. 2020. Curriculum learning for reinforcement learning domains: A framework and survey. *Journal of Machine Learning Research* 21(181): 1–50.

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.

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. 2020. Mastering atari, go, chess and shogi by planning with a learned model. *Nature* 588(7839): 604–609.

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. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362(6419): 1140–1144.

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

Taylor, M. E.; and Stone, P. 2009. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research* 10(Jul): 1633–1685.

Toro Icarte, R.; Klassen, T. Q.; Valenzano, R.; and McIlraith, S. A. 2018. Teaching multiple tasks to an RL agent using LTL. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, 452–461. International Foundation for Autonomous Agents and Multiagent Systems.

Toro Icarte, R.; Waldie, E.; Klassen, T.; Valenzano, R.; Castro, M.; and McIlraith, S. 2019. Learning reward machines for partially observable reinforcement learning. *Advances in Neural Information Processing Systems* 32: 15523–15534.

Weiss, K.; Khoshgoftaar, T. M.; and Wang, D. 2016. A survey of transfer learning. *Journal of Big data* 3(1): 1–40.

Xu, Z.; Gavran, I.; Ahmad, Y.; Majumdar, R.; Neider, D.; Topcu, U.; and Wu, B. 2020. Joint inference of reward machines and policies for reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 590–598.