Iterative Bounding MDPs: Learning Interpretable Policies via Non-Interpretable Methods

Nicholay Topin, Stephanie Milani, Fei Fang, Manuela Veloso
Carnegie Mellon University
Pittsburgh, PA 15213
{ntopin, smilani, feif, veloso}@cs.cmu.edu

Abstract

Current work in explainable reinforcement learning generally produces policies in the form of a decision tree over the state space. Such policies can be used for formal safety verification, agent behavior prediction, and manual inspection of important features. However, existing approaches fit a decision tree after training or use a custom learning procedure which is not compatible with new learning techniques, such as those which use neural networks. To address this limitation, we propose a novel Markov Decision Process (MDP) type for learning decision tree policies: Iterative Bounding MDPs (IBMDPs). An IBMDP is constructed around a base MDP so each IBMDP policy is guaranteed to correspond to a decision tree policy for the base MDP when using a method-agnostic masking procedure. Because of this decision tree equivalence, any function approximator can be used during training, including a neural network, while yielding a decision tree policy for the base MDP. We present the required masking procedure as well as a modified value update step which allows IBMDPs to be solved using existing algorithms. We apply this procedure to produce IBMDP variants of recent reinforcement learning methods. We empirically show the benefits of our approach by solving IBMDPs to produce decision tree policies for the base MDPs.

1 Introduction

The incorporation of deep neural networks into reinforcement learning (RL) has broadened the set of problems solvable with RL. Though these techniques yield high-performing agents, the policies are encoded using thousands to millions of parameters, and the parameters interact in complex, non-linear ways. As a result, directly inspecting and verifying the resulting policies is difficult. Without a mechanism for a human operator to readily inspect the resulting policy, we cannot deploy deep RL (DRL) in environments with strict regulatory or safety constraints.

Decision trees (DTs) (Quinlan 1986) are an interpretable model family commonly used to represent policies. Some benefits of DTs include that they allow formal verification of policy behavior (Bastani, Pu, and Solar-Lezama 2018), counterfactual analysis (Sokol and Flach 2019), and identification of relevant features. However, DRL techniques are not directly compatible with policies expressed as DTs. In contrast, some traditional RL algorithms, such as UTree (McCallum 1997), produce DT policies but use specific internal representations that cannot be replaced with a deep neural network. An alternative approach is to approximate a DRL policy with a DT, but the resulting policy can be arbitrarily worse than the original one, and the DT can be large due to unnecessary intricacies in the original policy.

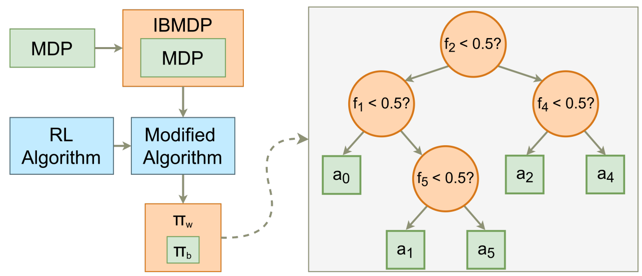

Figure 1: Method overview: we wrap a base MDP to form an IBMDP and solve using a modified RL algorithm. The solution is a decision tree policy for the base environment.

To address the limitations of these techniques, we propose to solve a meta-problem using RL such that the solution corresponds to a DT-format policy for the original problem. We introduce CUSTARD (Constrain Underlying Solution to a Tree; Apply RL to Domain), a process that uses RL to solve the meta-problem while ensuring that the embedded solution is equivalent to a DT-format policy throughout training (overview in Figure 1). We propose a novel Markov Decision Process (MDP) formulation for the meta-problem: Iterative Bounding MDPs. We present a general procedure to make RL techniques compatible with CUSTARD. CUSTARD allows modern DRL techniques to be applied to the meta-problem, since the learned policy weights can be fully replaced by an equivalent DT. Thus, CUSTARD maintains the interpretability advantage of a DT policy while using a non-interpretable function approximator during training. Additionally, CUSTARD ensures that the DT is an exact representation of the learned behavior, not an approximation.

The main contributions of this work are: (1) we introduce a novel MDP representation (IBMDPs) for learning a DT policy for a base MDP; (2) beginning with a two-agent UTree-like algorithm, we present an equivalent single-agent formulation which solves IBMDPs to produce DTs; (3) we show how to modify existing RL algorithms (policy gradient and Q-learning) to produce valid DTs for the base MDP; and (4) we empirically evaluate the performance of our approach and identify cases where it outperforms post-hoc DT fitting.

2 Background

2.1 Solving Markov Decision Processes

In the RL framework, an agent acts in an environment defined by an MDP. An MDP is a five-tuple $\langle S, A, T, R, \gamma \rangle$, consisting of a set of states S, a set of actions A, a transition function T, a reward function R, and a discount factor γ. We focus on factored MDPs (Boutilier et al. 1995), in which each state consists of a set of feature value assignments $s = \{f_1, ..., f_n\}$. Note that we do not require a factored reward function. An agent is tasked with finding a policy, $\pi : S \rightarrow A$, that yields the highest expected discounted future return for all states. The expected return for a state s when following policy π is $V^\pi(s) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t]$. Analogously, the expected return when taking action a and following π afterward is the Q-function $Q^\pi(s, a) = \mathbb{E}_\pi[r_0 + \gamma V^\pi(s')] = \mathbb{E}_\pi[r_0 + \gamma \max_{a'} Q^\pi(s', a')]$.

Q-learning-based algorithms directly approximate the Q-function, such as with a neural network (Mnih et al. 2013), and use it to infer the optimal policy π^*. The Q-function estimate is incrementally updated to be closer to a *target*, the bootstrapped estimate $r_t + \gamma \max_{a'} Q(s_{t+1}, a')$. In contrast, policy gradient methods directly model and optimize the policy. Actor-critic methods (Schulman et al. 2017) additionally model the value function to leverage it in the policy update. They often use a critic for estimating the advantage function, $A(s, a) = Q(s, a) - V(s)$.

2.2 Decision Trees in Reinforcement Learning

Decision Trees (DTs) recursively split the input space along a specific feature based on a cutoff value, yielding axis-parallel partitions. *Leaf nodes* are the final partitions; *internal nodes* are the intermediate partitions. DT-like models have been used to represent the transition model (Strehl, Diuk, and Littman 2007), reward function (Degris, Sigaud, and Wuillemin 2006), value function (Pyeatt and Howe 2001; Tuyls, Maes, and Manderick 2002), relative effect of actions (Hester, Quinlan, and Stone 2010), and policy (McCallum 1997). We focus on DT policies (DTPs), which map each state to a leaf node representing an action.

Sufficiently small DTPs are *interpretable* (Molnar 2019), in that people understand the mechanisms by which they work. DTs conditionally exhibit simulatability, decomposability, and algorithmic transparency (Lipton 2018). When a person can contemplate an entire model at once, it is *simulatable*; sufficiently small DTs exhibit this property. A *decomposable* model is one in which sub-parts can be intuitively explained; a DT with interpretable inputs exhibits this property. *Algorithmic transparency* requires an understanding of the algorithm itself; DTs are verifiable (Bastani, Pu, and Solar-Lezama 2018), which is important in safety-critical applications.

3 Related Work

3.1 Decision Tree Policies

Prior work on creating DTPs using arbitrary function approximators focuses on explaining an existing agent; a non-DT policy is learned, then approximated using a DT. One such method is VIPER (Bastani, Pu, and Solar-Lezama 2018), which uses model compression techniques (Bucilu, Caruana, and Niculescu-Mizil 2006; Hinton, Vinyals, and Dean 2015; Rusu et al. 2015) to distill a policy into a DT. This work adapts DAGGER (Ross, Gordon, and Bagnell 2011) to prioritize gathering critical states, which are then used to learn a DT. MOET (Vasic et al. 2019) extends VIPER by learning a mixture of DTPs trained on different regions of the state space. However, both VIPER and MOET approximate an expert. When the expert is poorly approximated by a DT, the resulting DTPs perform poorly.

Other lines of work focus on directly learning a DTP, but they cannot use an arbitrary function approximator. UTree (McCallum 1997) and its extensions (Uther and Veloso 1998; Pyeatt 2003; Roth et al. 2019) incrementally build a DT while training an RL agent. Transition tuples (or tuple statistics) are stored within leaf nodes, and a leaf node is split when the tuples suggest that two leaf nodes would better represent the Q-function. A concurrent work (Rodriguez et al. 2020) uses a differential decision tree to represent the policy and approximates the soft-max tree with a DT after training. However, these methods require specific policy or Q-function representations, so they cannot leverage powerful function approximators like neural networks.

3.2 Interpretable Reinforcement Learning

There exist other methods to produce interpretable RL policy summarizations. One line of work produces graph-structured policies, including finite state machines (Koul, Greydanus, and Fern 2018), Markov chains (Topin and Veloso 2019), and transition graphs between landmarks (Sreedharan, Srivastava, and Kambhampati 2020). Other work produces policies in custom representations. For example, Hein, Udluft, and Runkler (2019) use a genetic algorithm to create policy function trees, which have algebraic functions as internal nodes and constants and state variables as leaves. Hein et al. (2017) express a policy as a fuzzy controller, which is a set of linguistic if-then rules whose outputs are combined. These policy formats address different aspects of interpretability compared to DTPs (e.g., showing long-term behavior rather than allowing policy verification).

Another line of work uses attention mechanisms (Wang et al. 2018; Annasamy and Sycara 2019; Tang, Nguyen, and Ha 2020) to determine crucial factors in individual decisions made by the agent. A similar line of work is saliency methods, which produce visual representations of important pixels (Greydanus et al. 2018; Yang et al. 2018; Huber, Schiller, and André 2019) or objects (Iyer et al. 2018). However, Atrey, Clary, and Jensen (2020) argue that saliency maps are not sufficient explanations because the conclusions drawn from their outputs are highly subjective. Other methods explain decisions made by the agent as a function of the MDP components or the training process, including the reward function (Anderson et al. 2019; Juozapaitis et al. 2019; Tabrez, Agrawal, and Hayes 2019), transition probabilities (Cruz, Dazeley, and Vamplew 2019), and causal relationships in the environment (Madumal et al. 2020a,b). These methods are orthogonal to our work; they provide different insights and can be used alongside our approach.

3.3 Hierarchical Reinforcement Learning

Our method can be viewed as a type of hierarchical decomposition, similar to that performed in hierarchical RL (Dayan and Hinton 1993; Dietterich 2000). Perhaps the most well-known formulation is the options framework (Precup, Sutton, and Singh 1998; Sutton, Precup, and Singh 1999), in which the problem is decomposed into a two-level hierarchy. At the bottom level are options, which are subpolicies with termination conditions that take observations of the environment as input and output actions until the termination condition is met. A policy is defined over options; using this policy, an agent chooses an option, then follows it until termination. Upon termination, the policy over options is again queried, and so on. Options over an MDP define a semi-MDP (Bradtke and Duff 1994; Mahadevan et al. 1997; Parr and Russell 1998). In our method, the base MDP can be viewed as this semi-MDP and the IBMDP can be viewed as the full MDP. In a sense, the policy for the information-gathering actions is the lower-level policy, and the higher-level policy selects over the information-gathering policies.

4 Approach

We present CUSTARD, an approach for training an agent to produce a DT policy using existing RL algorithms. We achieve this goal by training the agent to solve a wrapped version of the original, *base* MDP. The wrapped MDP, which we name an *Iterative Bounding MDP*, extends the base MDP by adding information-gathering actions and *bounding* state features to indicate the gathered information. The new features correspond to a position during a DT traversal, and the new actions correspond to partitions performed by internal nodes in a DT. By constraining an agent's policy to be a function of the bounding state features, the learned policy is equivalent to a DT.

In Section 4.1, we describe IBMDPs. In Section 4.2, we describe the process for extracting a DTP from an IBMDP policy. In Section 4.3, we present methods for adapting existing RL algorithms to find DTPs by solving IBMDPs.

4.1 Iterative Bounding MDPs

We introduce Iterative Bounding MDPs (IBMDPs), a novel MDP formulation for producing DTPs. We seek to produce a DTP by ensuring that an agent's IBMDP policy is equivalent to a DTP for the original, base MDP.

The base MDP must be an MDP with a factored state representation, where each state feature has upper and lower bounds on its values. A *base state* is a state from the base MDP's state space, and a *wrapped state* is a state from the IBMDP's state space; other terms are defined analogously.

State Space A wrapped state s_w consists of two parts: a base state s_b and bounding features, f^l_1-f^l_n and f^h_1-f^h_n. There exist two bounding features per base state feature, such that f^l_i represents a lower bound on the base feature f_i's current value, and f^h_i represents an upper-bound for that same base feature's current value. The bounding features reflect the outcomes of binary comparisons performed during the traversal, and the bounds are tightened with more comparisons. A sequence of wrapped states represents a traversal through a DTP for a specific s_b. For simplicity, and without loss of generality, we consider s_b to be normalized such that all features are in $[0, 1]$. We use $s_w[c]$ to refer to a component c within s_w. An IBMDP state and state space are:

$$S_w = S_b \times [0,1]^{2n}, \;\; s_w = \langle s_b, f^l_1, \ldots, f^l_n, f^h_1, \ldots, f^h_n \rangle.$$

Action Space An IBMDP's action space A_w consists of base actions A_b and information-gathering actions A_I:

$$A_w = A_b \cup A_I.$$

Base actions correspond to taking an action within the base MDP, as when reaching a leaf in a DTP. Information-gathering actions specify a base state feature and a value, which correspond to the feature and value specified by an internal node of a DTP. We present two different action space formats: a discrete set of actions and a Parameterized Action Space (Masson, Ranchod, and Konidaris 2016). In both cases, the action can be described by a tuple, $\langle c, v \rangle$, where c is the chosen feature and v is the value. For simplicity, we consider $v \in [0, 1]$, where 0 and 1 respectively correspond to the current lower and upper bound on c.

With a discrete set of IBMDP actions, each of the n features can be compared to one of p possible values. This results in $p \times n$ discrete actions, with v values of $1/(p+1), \ldots, p/(p+1)$ for each of the n possible f. With this construction, the base actions must be discrete. In this case, the information-gathering actions are:

$$A_I = \{c_1, \ldots, c_n\} \times \left\{ \frac{1}{p+1}, \ldots, \frac{p}{p+1} \right\}.$$

In a Parameterized Action Space MDP (PASMDP), each action $a \in A_d$ has m_a continuous parameters. A specific action choice is specified by selecting $(a, p^a_1, \ldots, p^a_{m_a})$. If the IBMDP is a PASMDP, then there is an action for each of the n features with a single parameter ($m_a = 1$), where the action specifies c and the parameter specifies v. With this formulation, the base MDP may have a continuous, multi-dimensional action space. This is supported by adding a single a with parameters corresponding to the base action choices. If A_b has discrete actions, then an a is added for each of them, with the corresponding m_a set to zero. The information-gathering actions in the PASMDP variant are:

$$A_I = \{c_1, \ldots, c_n\} \times [0, 1].$$

Transition Function When an agent takes an information-gathering action, $\langle c, v \rangle$, the selected value v is compared to the indicated feature c. Since v is constrained to $[0,1]$ but represents values in $[c^l, c^h]$, the un-normalized v_p is obtained by projecting $v_p \leftarrow v \times (c^h - c^l) + c^l$. The bounding features c^l and c^h are updated to reflect the new upper- and lower-bounds for c; the base features are unchanged. This process is equivalent to the behavior of an internal node in a DTP: a feature is compared to a value, and the two child nodes represent different value ranges for that feature. Thus, for an information-gathering action $\langle c, v\rangle$, the transition function of the IBMDP, T_w, is deterministic, and the next state, s'_w, is based on s_w:

$$s'_w[s_b] = s_w[s_b],$$
$$s'_w[f] = s_w[f] \forall f \notin \{c^l, c^h\},$$
$$\text{If } s_b[c] \leq v_p\text{: } s'_w[c^h] = \min(s_w[c^h], v_p), s'_w[c^l] = s_w[c^l],$$
$$\text{If } s_b[c] > v_p\text{: } s'_w[c^l] = \max(s_w[c^l], v_p), s'_w[c^h] = s_w[c^h].$$

When a base action is taken, the base features are updated as though this action was taken in the base MDP, and the bounding features are reset to their extreme values. This is equivalent to selecting a base action in a DTP and beginning to traverse the DTP for the next base state (starting from the root node). This corresponds to a transition function of:

$$a \in A_b \wedge ((s'_w[f^l_i] = 0) \wedge (s'_w[f^h_i] = 1) \forall i \in \{1, \ldots, n\})$$
$$\implies T_w(s_w, a, s'_w) = T_b(s_w[s_b], a, s'_w[s_b]).$$

Reward Function The reward for a base action is the reward specified by the base MDP for the base action, base original state, and base new state. The reward for information-gathering actions is a fixed, small penalty ζ. For a sufficiently low value of ζ, the optimal solution for the IBMDP includes the optimal solution of the base MDP. The overall IBMDP reward function is:

$$a \in A_b \implies R(s_w, a, s'_w) = R(s_w[s_b], a, s'_w[s'_b]),$$
$$a \notin A_b \implies R(s_w, \langle c, v\rangle, s'_w) = \zeta.$$

Gamma We introduce a second discount factor, γ_w. When a base action is taken in the IBMDP, the gamma from the base MDP, γ_b, is used to compute the expected discounted future reward. Otherwise, γ_w is used. For a γ_w sufficiently close to 1, the expected discounted future reward is identical for an s_w, if acted upon in the IBMDP, and its corresponding s_b, if acted upon in the base MDP.

Remaining Components We present the additional components required for an episodic MDP, but the framework is also applicable to non-episodic environments. A transition in the IBMDP, (s_w, a_w, s'_w), is terminal if $a \in A_b$ and $(s_w[s_b], a, s'_w[s_b])$ is a terminal transition in the base MDP. The distribution over starting states of the IBMDP is derived from the distribution of starting states in the base MDP.

4.2 Tree Extraction

Not all policies for the IBMDP correspond to valid DTPs; the presence of s_b within each wrapped state allows access to full state information at any point during tree traversal. However, all IBMDP policies that only consider the bounding features (i.e., ignore s_b) correspond to a DTP. We describe the process for extracting a DTP from a policy defined over bounding observations from the environment, $\pi(s_w \setminus s_b)$. We present the training of such policies in Section 4.3.

Algorithm 1 Extract a Decision Tree Policy from an IBMDP policy π, beginning traversal from obs.

1: **procedure** SUBTREE_FROM_POLICY(obs, π)
2: $\quad a \leftarrow \pi(obs)$
3: $\quad$ **if** $a \in A_b$ **then** $\quad \triangleright$ Leaf if base action
4: $\quad\quad$ **return** Leaf_Node(action: a)
5: $\quad$ **else**
6: $\quad\quad c, v \leftarrow a \quad \triangleright$ Splitting action is feature and value
7: $\quad\quad v_p \leftarrow v \times (obs[c^h] - obs[c^l]) + obs[c^l]$
8: $\quad\quad obs_L \leftarrow obs; \quad obs_R \leftarrow obs$
9: $\quad\quad obs_L[c^h] \leftarrow v_p; \quad obs_R[c^l] \leftarrow v_p$
10: $\quad\quad child_L \leftarrow$ Subtree_From_Policy(obs_L, π)
11: $\quad\quad child_R \leftarrow$ Subtree_From_Policy(obs_R, π)
12: $\quad\quad$ **return** Internal_Node(feature: c, value: v_p, children: $(child_L, child_R)$)

Algorithm 1 outlines the DTP extraction procedure. SUBTREE_FROM_POLICY constructs a single node based on the current observation. Recursive calls are used to contstruct the children. The bounding features $(s_w \setminus s_b)$ describe a node within a DTP, with $s_w[f^l_i] = 0 \wedge s_w[f^h_i] = 1 \forall i \in [1, \ldots, n]$ corresponding to the root node. SUBTREE_FROM_POLICY($s_w \setminus s_b, \pi$) for a root node s_w yields the DTP for π.

An action a within the IBMDP corresponds to a leaf node action (when $a \in A_b$) or a DT split (when $a \notin A_b$). Lines 2-3 obtain the current action and identify its type. Line 4 constructs a leaf if a is not an information gathering action. Information gathering actions consist of a feature choice c and a splitting value v (Line 6). The IBMDP constrains v to be in $[0,1]$, which corresponds to decision node splitting values between $s_w[c^l]$ and $s_w[c^h]$, the current known upper and lower bounds for feature c. Line 7 projects v onto this range, yielding v_p, to which feature c can be directly compared.

To create the full tree, both child nodes must be explored, so the procedure considers both possibilities ($s_b[c] \leq v_p$ and $s_b[c] > v_p$). Lines 8-9 construct both possible outcomes: a tighter upper bound, $c^h \leftarrow v_p$, and a tighter lower bound, $c^l \leftarrow v_p$. This procedure then recursively creates the child nodes (Lines 10-11). The final result (Line 12) is an internal DTP node: an incoming observation's feature is compared to a value v_p ($obs[c] \leq v_p$), and traversal continues to one of the children, depending on the outcome of the comparison.

4.3 Training Procedure

If an agent solves an IBMDP without further constraints, then it can learn a policy where actions depend on s_b in arbitrarily complicated ways. To ensure that the base MDP policy follows a DT structure, the IBMDP policy must be a function of only the bounding features. Effectively, if the policy is a function of $s_w \setminus s_b$, then the policy is a DTP for the

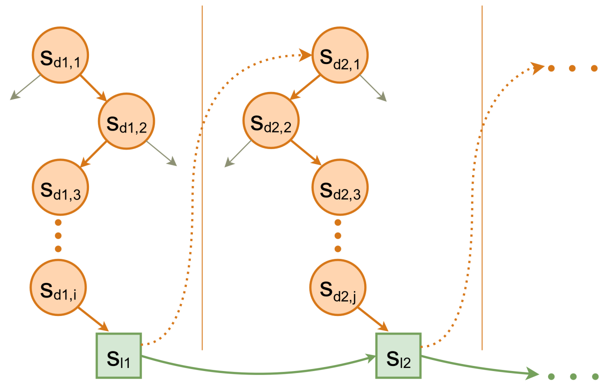

Figure 2: The division between the tree agent (orange circle states and arrow actions) and the leaf agent (green square states and arrow actions). Each tree traversal is an episode for the tree agent and one transition for the leaf agent.

base MDP. However, with a policy of the form $\pi(s_w \setminus s_b)$, the standard bootstrap estimate does not reflect expected future reward because the next observation is always the zero-information root node state. Therefore, standard RL algorithms must be modified to produce DTPs within an IBMDP.

We present a set of modifications that can be applied to standard RL algorithms so the one-step bootstrap reflects a correct future reward estimate. We motivate this set of modifications by presenting a "two agent" division of the problem and then show the equivalent single-agent Q target. We then demonstrate how a target Q-function or critic can be provided the full state (s_w) to facilitate learning while maintaining a DT-style policy. Finally, we present how the modifications are applied to Q-learning and actor-critic algorithms. Without loss of generality, we focus on learning a Q-function. If learning an advantage function or value function, an analogous target modification can be made.

Two Agent Division Learning in an IBMDP can be cast as a two-agent problem: (i) a *tree agent* selects which information-gathering actions to take and when to take a base action, and (ii) a *leaf agent* selects a base action using the bounding features, when prompted to do so. Figure 2 shows this division, where the leaf agent selects actions in s_{l_1} and s_{l_2}, and the tree agent selects all other actions.

With this division of the problem, the leaf agent is equivalent to the agent in UTree-style methods. The tree agent replaces the incremental tree construction used in UTree and is akin to an RL agent constructing a DT for a supervised problem (Preda 2007). The leaf agent's observed transition sequence consists of leaf nodes and its own selected actions: $s_{l_1}, a_{l_1}, r_{l_1}, s_{l_2}, a_{l_2}$. The bootstrapped Q-value estimate is:

$$r_{l_1} + \gamma_b \max_{a' \in A_b} Q_l(s_{l_2}, a'),$$

where r_{l_1} is a reward obtained from the base MDP.

In this framing, the tree agent experiences a new episode when a base action is taken. The initial state is always the zero-information, root state, and the episode terminates when the agent chooses the stop splitting action, a_{stop}, which we add for the two agent formulation. When the tree agent stops splitting, the reward is the value estimated by the leaf agent, $Q_l(s_l, a_l)$. The tree agent's Q-value target is:

$$r_d + \gamma_w \max_{a' \in a_{stop} \cup A_w \setminus A_b} Q_d(s'_d, a'),$$

where r_d is $\max_{a' \in A_b} Q_l(s_d, a')$ if the a_{stop} action was chosen and ζ otherwise. When a_{stop} is taken, $Q_d(s'_d, a')$ is 0 for all a' since the transition is terminal for the tree agent.

These two equations for target Q-values allow an IBMDP to be solved using only the partial $s_w \setminus s_b$ observations. The tree agent does not directly receive a reward signal from future base actions but uses the leaf agent's estimates to update. The leaf agent learns long-term reward estimates based on rewards from the environment.

Merging of Agents The target Q-value for a terminal tree agent action is r_d, which is $\max_{a \in A_b} Q_l(s, a)$. The tree agent's episode terminates if and only if a_{stop} is taken. Effectively, the tree agent seeks to learn $Q_d(s, a_{stop}) = \max_{a \in A_b} Q_l(s, a)$. Rather than learning this relationship, $Q_d(s, a_{stop})$ can directly query Q_l, simplifying the learning task without changing the underlying problem.

With this change to $Q_d(s, a_{stop})$, Q_d and Q_l are defined over disjoint subsets of A_w. A single, unified Q-function Q can be learned, defined over all a in A_w, with targets:

$$a \in A_b \implies \text{target} = r_{l_1} + \gamma_b \max_{a' \in A_b} Q(s_{l_2}, a'),$$
$$a \notin A_b \implies \text{target} = \zeta + \gamma_w \max_{a' \in A_w} Q(s', a'),$$

where s' is the next state, regardless of type. In the former equation, s_{l_2} is the next state in which a base action is taken when following the greedy policy. In the latter equation, if the max returns the Q-value for an $a \notin A_b$, the two terms correspond to the reward and expected discounted future reward. When the max returns the Q-value for an $a \in A_b$, the two terms are then the immediate reward and the reward from a_{stop} in the next state, effectively removing the terminal/non-terminal distinction for the tree agent.

As a result, this two-agent problem is equivalent to a single agent updating a single Q-function using two different targets, depending on the action taken. The equation for computing a target differs from the standard Q-function update equation (as applied to the IBMDP) in one way: if a base action is taken, the "next state" is the next state in which a base action is taken, rather than simply the next state. This single change is sufficient to learn DTPs for IBMDPs.

Omniscient Q-function The merged agent formulation requires the next leaf state, s_{l_2}, when a base action is taken. This state is not naturally encountered when performing off-policy exploration, so s_{l_2} must be computed by repeatedly querying the Q-function with a sequence of s_d tree states until the next base action is chosen. This simulation of the next choice increases the computation for a learning update.

We propose to approximate $Q(s_{l_2}, a)$ using a second Q-function, Q_o. We refer to it as the *omniscient Q-function*

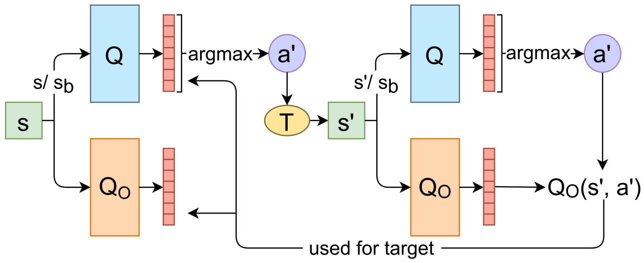

Figure 3: The method for using the omniscient Q-function, Q_o, for Q targets. The policy is based only on Q, so a DTP can be extracted despite Q_o being a function on the full state.

since its input is the full state s_w. Q_o is used only during training; the policy is obtained directly from Q. As a result, providing Q_o the full state does not violate the requirement that the policy is a function of only $s_w \setminus s_b$.

The omniscient Q-function is trained to approximate $Q(s_{l_2}, a)$ using a and the full state at s_{l_2}'s root, s_r. This root state is sufficient since s_{l_2} is obtained from s_r through a sequence of actions, each based on the previous s_d. Therefore, the current greedy policy corresponds to some function $F(s_r) = s_{l_2}$ for all (s_r, s_{l_2}) pairs. Q_o implicitly learns $F(s_r)$ as it learns $Q_o(s_r, a) \approx Q(s_{l_2}, a)$ for all base actions.

The original merged formulation learns the Q-value at each level in the tree using targets computed from the next level. This leads to slow propagation of environment reward signals from the leaf nodes. We propose to have Q_o learn to approximate $Q_o(s_w, a) \approx Q(s, a)$ for all states and all actions. Since Q_o has access to s_b, the rewards obtained in the leaf node, s_{l_1}, propagate through Q_o to earlier levels of the tree instead of sequentially propagating from leaf to root.

As shown in Figure 3, during training, we use Q_o in cases where $Q(s, a)$ would be used as a target. The action choice is still based on $Q(s, a)$, but the value is obtained from Q_o. Both Q_o and Q are updated using the Q_o-based target.

Modifying Standard RL Algorithms For Q-learning-based methods, such as Dueling Deep Q-Networks (DDQN) (Wang et al. 2016) and Model-Free Episodic Control (MFEC) (Blundell et al. 2016), we use the merged agent formulation target value of $Q_o(s_w, \arg\max_a Q(s_w \setminus s_b, a))$ in place of $\max_a Q(s, a)$ (for both $a \in A_b$ and $a \notin A_b$); Q_o is updated using the same target value. For policy gradient methods, such as Proximal Policy Optimization (PPO) (Schulman et al. 2017), the critic is used to compute advantage values only during training. Therefore, we use a Q_o-style critic to compute advantages, which is trained using the merged agent formulation target value computations.

5 Experiments

We evaluate CUSTARD's ability to create DTPs using a non-interpretable function approximator during the learning process. An alternative approach is to learn a non-tree expert policy and then find a tree that mimics the expert. We compare to VIPER, which takes this alternative approach and outperforms standard imitation learning methods. We evaluate with three environments, briefly described in Section 5.1. Further environment details and experiment parameters are in the Appendix (available at arxiv.org/abs/2102.13045).

5.1 Environments

CartPole (Barto, Sutton, and Anderson 1983) The agent balances a pole by moving laterally. We use the OpenAI Gym (Brockman et al. 2016) variant: An episode terminates when the pole falls or 200 timesteps have elapsed. Following previous work, methods are limited to DTs of depth two.

PrereqWorld (Topin and Veloso 2019) The agent is tasked with creating a goal item, and creating an item may require a subset of other, prerequisite items. This environment is similar in structure to advising domains (Dodson, Mattei, and Goldsmith 2011) and crafting, as in MineRL (Milani et al. 2020). The environment has a number of item types m which corresponds to the number of actions and binary state features. We use a base environment with $m = 10$ and a fixed prerequisite hierarchy. We produce smaller variants by removing items lower in the hierarchy.

PotholeWorld We introduce a new domain, where the agent switches lanes to avoid potholes. The first lane gives less reward per unit traveled, but the other two lanes contain sporadic "potholes" which lead to a larger reward penalty if traversed. A state contains a single feature: the current position. The three actions are $[lane\_1, lane\_2, lane\_3]$, which each advance the agent a random distance.

5.2 Learning with CUSTARD

To evaluate CUSTARD's ability to produce DTPs with a non-interpretable function approximator for the IBMDP, we apply the CUSTARD modifications to three base methods: DDQN, PPO, and MFEC with improvements from Neural Episodic Control (Pritzel et al. 2017). DDQN uses Q-learning and a neural network to learn state-value and action-value functions. PPO is a policy gradient method that uses a critic to learn the advantage function. We use a neural network for both the actor and critic. MFEC uses Q-learning and estimates Q-values with a nearest neighbors model.

The three methods are modified as described in Section 4.3. Actions are selected based on the bounding features $(s_w \setminus s_b)$; this affects the actor for PPO and the Q-function for DDQN and MFEC. DDQN and MFEC are used with a target Q-value function, Q_o, when performing updates, as in Figure 3. Q_o and the critic for PPO use the full IBMDP state.

We compare to VIPER using two expert types: DQN and Backward Induction (BI). In Table 1, we show the final average reward and tree depth for 50 trials on CartPole, PrereqWorld ($m = 7$), and PotholeWorld. Optimal final average rewards would be 200, -4, and 50, respectively. CUSTARD finds DTPs with high average reward for all environments and tends to find shorter DTPs than VIPER. To further evaluate depth-vs.-reward trade-offs, we use VIPER(BI) and CUSTARD(MFEC) since these methods have the fewest hyperparameters and are least computationally expensive.

	CartPole	PrereqWorld		PotholeWorld	
	Rew.	Rew.	Dep.	Rew.	Dep.
VIPER (DQN)	200.00	-4.00	5.70	46.30	6.00
VIPER (BI)	200.00	-4.00	6.00	46.31	9.18
Ours (DQN)	198.72	-4.08	4.28	46.92	5.36
Ours (PPO)	199.32	-4.04	4.16	45.39	1.04
Ours (MFEC)	200.00	-4.00	3.92	49.18	9.74

Table 1: Final average cumulative reward and tree depth.

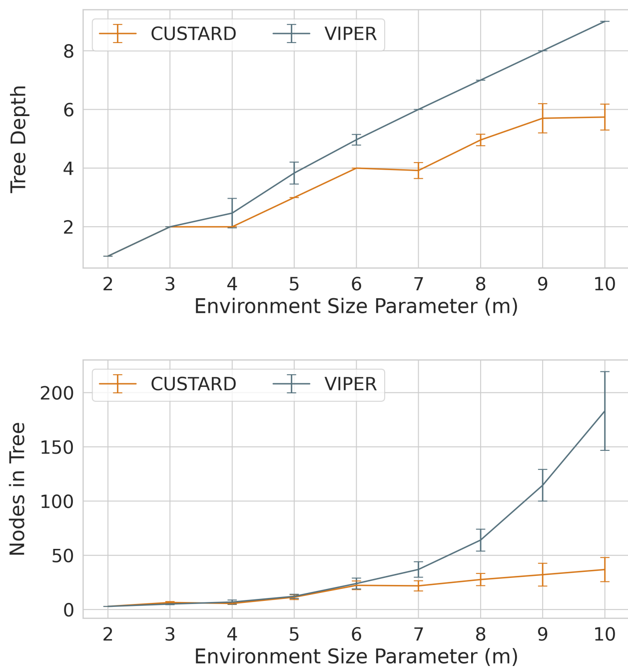

Figure 4: Tree depth and node count as the PrereqWorld environment size increases (Std Dev bars). CUSTARD yields smaller trees for larger environments than VIPER.

5.3 Response to Environment Size

CUSTARD discourages the learning of unnecessarily large trees through the use of two penalty terms, ζ and γ_w. These penalties are akin to regularization of the implicit DTP: when multiple optimal DTPs exist for the base MDP, the optimal IBMDP policy corresponds to the DTP with the lowest average leaf height. In contrast, if a tree mimics an expert policy, then the resulting tree will include complex behaviors that are artifacts of the expert's intricacy.

We evaluate the decrease in tree size attained by using CUSTARD to directly learn a tree. We compare the tree depth and node count for DTPs found by VIPER and CUSTARD on PrereqWorld. The environment size is varied through m, which specifies the number of states (2^m) and the number of actions (m). For a given m, average reward is equal for both methods. The results are shown in Figure 4 (50 trials per method/m pair). CUSTARD produces smaller trees for $m \geq 4$, and the size differences increases with the environment size. This is because an unconstrained expert can learn more complex behaviors with a larger state space, and VIPER faithfully mimics the expert policy.

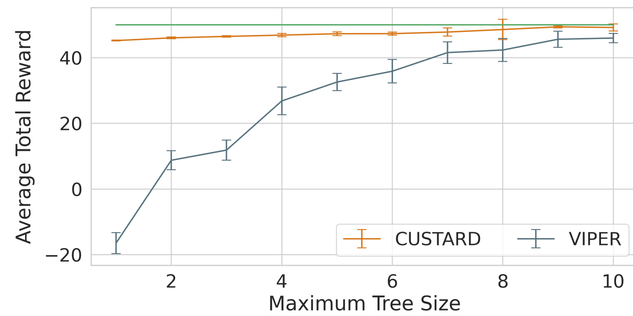

Figure 5: Average per-episode reward for trees of a fixed depth for PotholeWorld (Std Dev bars, max reward line at 50). CUSTARD produces DTPs that achieve higher reward.

5.4 Response to Tree Depth

If an application requires a DTP of fixed depth, then fitting a DT to an expert policy can yield a poor policy of that depth. This is because the expert is not learned in the context of the depth limitation; imperfectly imitating that expert can lead to low reward. CUSTARD yields better policies at a given depth since it directly solves an IBMDP that can be augmented with a depth limit. An IBMDP can include affordances (Khetarpal et al. 2020), so that information-gathering actions cannot be chosen n actions after the most recent base action. With this modification, an RL algorithm can directly find the best DTP subject to the depth restriction.

We evaluate CUSTARD's ability to find DTPs with high average reward for PotholeWorld subject to a tree depth limit. This domain is designed so the overall optimal DTP cannot be pruned to obtain the optimal DTP for a smaller depth. We present the average episode reward as a function of the depth limit in Figure 5 for VIPER and CUSTARD (50 trials per method/depth pair). CUSTARD attains higher reward through using $lane\_1$ when the DTP depth is too shallow to avoid potholes in the other lanes. In contrast, VIPER always attempts to imitate the expert and attains a low reward when the DTP poorly represents the expert policy.

6 Conclusion and Future Work

We introduce Iterative Bounding MDPs, an MDP representation which corresponds to the problem of finding a decision tree policy for an underlying MDP. Additionally, we identify how the standard value update rule must be changed so all IBMDP solutions correspond to decision tree policies for the underlying MDP. We show how to modify existing RL algorithms to solve IBMDPs, so these methods can produce decision tree policies when using non-interpretable function approximators during training. We provide empirical results showing the tree size and reward improvements possible through solving an IBMDP rather than approximating a non-interpretable expert. Future work includes generalization of IBMDPs to encode other constraints.

Acknowledgements

This material is based upon work supported by the Department of Defense (DoD) through the National Defense Science & Engineering Graduate (NDSEG) Fellowship Program, DARPA/AFRL agreement FA87501720152, and NSF grant IIS-1850477. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Department of Defense, Defense Advanced Research Projects Agency, the Air Force Research Laboratory, or the National Science Foundation.

References

Anderson, A.; Dodge, J.; Sadarangani, A.; Juozapaitis, Z.; Newman, E.; Irvine, J.; Chattopadhyay, S.; Fern, A.; and Burnett, M. 2019. Explaining reinforcement learning to mere mortals: An empirical study. In *Proc. of the 28th Int. Joint Conf. on Artificial Intelligence*.

Annasamy, R. M.; and Sycara, K. 2019. Towards better interpretability in deep q-networks. In *Proc. of the 33rd AAAI Conf. on Artificial Intelligence*.

Atrey, A.; Clary, K.; and Jensen, D. 2020. Exploratory Not Explanatory: Counterfactual Analysis of Saliency Maps for Deep RL. In *Proc. of the 8th Int. Conf. on Learning Representations*.

Barto, A. G.; Sutton, R. S.; and Anderson, C. W. 1983. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics* .

Bastani, O.; Pu, Y.; and Solar-Lezama, A. 2018. Verifiable Reinforcement Learning via Policy Extraction. In *Advances in Neural Information Processing Systems*.

Blundell, C.; Uria, B.; Pritzel, A.; Li, Y.; Ruderman, A.; Leibo, J. Z.; Rae, J.; Wierstra, D.; and Hassabis, D. 2016. Model-free episodic control. *arXiv preprint arXiv:1606.04460* .

Boutilier, C.; Dearden, R.; Goldszmidt, M.; et al. 1995. Exploiting structure in policy construction. In *Proc. of the 14th Int. Joint Conf. on Artificial Intelligence*.

Bradtke, S.; and Duff, M. 1994. Reinforcement learning methods for continuous-time Markov decision problems. *Advances in Neural Information Processing Systems* .

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI Gym. *arXiv preprint arXiv:1606.01540* .

Bucilu, C.; Caruana, R.; and Niculescu-Mizil, A. 2006. Model compression. In *Proc. of the 12th ACM Int. Conf. on Knowledge Discovery and Data Mining*.

Cruz, F.; Dazeley, R.; and Vamplew, P. 2019. Memory-Based Explainable Reinforcement Learning. *AI 2019: Advances in Artificial Intelligence* .

Dayan, P.; and Hinton, G. E. 1993. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems*.

Degris, T.; Sigaud, O.; and Wuillemin, P.-H. 2006. Learning the structure of factored Markov decision processes in reinforcement learning problems. In *Proc. of the 23rd Int. Conf. on Machine Learning*.

Dietterich, T. G. 2000. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research* 13.

Dodson, T.; Mattei, N.; and Goldsmith, J. 2011. A natural language argumentation interface for explanation generation in Markov decision processes. In *Proc. of the 2nd Int. Conf. on Algorithmic Decision Theory*.

Greydanus, S.; Koul, A.; Dodge, J.; and Fern, A. 2018. Visualizing and Understanding Atari Agents. In *Proc. of the 35th Int. Conf. on Machine Learning*.

Hein, D.; Hentschel, A.; Runkler, T.; and Udluft, S. 2017. Particle swarm optimization for generating interpretable fuzzy reinforcement learning policies. *Eng. Applications of Artificial Intelligence* 65.

Hein, D.; Udluft, S.; and Runkler, T. A. 2019. Interpretable Policies for Reinforcement Learning by Genetic Programming. In *Proc. of the Genetic and Evolutionary Computation Conf.*

Hester, T.; Quinlan, M.; and Stone, P. 2010. Generalized model learning for reinforcement learning on a humanoid robot. In *Proc. of the Int. Conf. on Robotics and Automation*.

Hinton, G.; Vinyals, O.; and Dean, J. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* .

Huber, T.; Schiller, D.; and André, E. 2019. Enhancing explainability of deep reinforcement learning through selective layer-wise relevance propagation. In *Proc. of the Joint German/Austrian Conf. on Artificial Intelligence (Künstliche Intelligenz)*.

Iyer, R.; Li, Y.; Li, H.; Lewis, M.; Sundar, R.; and Sycara, K. 2018. Transparency and Explanation in Deep Reinforcement Learning Neural Networks. In *Proc. of the 1st AAAI/ACM Conf. on Artificial Intelligence, Ethics, and Society*.

Juozapaitis, Z.; Koul, A.; Fern, A.; Erwig, M.; and Doshi-Velez, F. 2019. Explainable reinforcement learning via reward decomposition. In *28th Int. Joint Conf. on Artificial Intelligence Workshop on Explainable Artificial Intelligence*.

Khetarpal, K.; Ahmed, Z.; Comanici, G.; Abel, D.; and Precup, D. 2020. What can I do here? A Theory of Affordances in Reinforcement Learning. *arXiv preprint arXiv:2006.15085* .

Koul, A.; Greydanus, S.; and Fern, A. 2018. Learning finite state representations of recurrent policy networks. *arXiv preprint, arXiv:1811.12530* .

Lipton, Z. C. 2018. The mythos of model interpretability. *ACM Queue* 16(3).

Madumal, P.; Miller, T.; Sonenberg, L.; and Vetere, F. 2020a. Distal Explanations for Explainable Reinforcement Learning Agents. *arXiv preprint arXiv:2001.10284* .

Madumal, P.; Miller, T.; Sonenberg, L.; and Vetere, F. 2020b. Explainable reinforcement learning through a causal lens. In *Proc. of the 34th AAAI Conf. on Artificial Intelligence*.

Mahadevan, S.; Marchalleck, N.; Das, T. K.; and Gosavi, A. 1997. Self-improving factory simulation using continuous-time average-reward reinforcement learning. In *Proc. of the 14th Int. Conf. on Machine Learning*.

Masson, W.; Ranchod, P.; and Konidaris, G. 2016. Reinforcement Learning with Parameterized Actions. In *Proc. of the 30th AAAI Conf. on Artificial Intelligence*.

McCallum, R. 1997. Reinforcement learning with selective perception and hidden state. *PhD Thesis, University of Rochester, Department of Computer Science* .

Milani, S.; Topin, N.; Houghton, B.; Guss, W. H.; Mohanty, S. P.; Nakata, K.; Vinyals, O.; and Kuno, N. S. 2020. Retrospective Analysis of the 2019 MineRL Competition on Sample-Efficient Reinforcement Learning Using Human Priors. *NeurIPS2019 Competition & Demonstration Track Postproceedings* .

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* .

Molnar, C. 2019. *Interpretable Machine Learning*. https://christophm.github.io/interpretable-ml-book/ (July 2020).

Parr, R. E.; and Russell, S. 1998. *Hierarchical control and learning for Markov decision processes*. University of California, Berkeley Berkeley, CA.

Precup, D.; Sutton, R. S.; and Singh, S. 1998. Theoretical results on reinforcement learning with temporally abstract options. In *European Conf. on Machine Learning*.

Preda, M. 2007. Adaptive Building of Decision Trees by Reinforcement Learning. In *Proc. of the 7th Int. Conf. on Applied Informatics and Communications*.

Pritzel, A.; Uria, B.; Srinivasan, S.; Puigdomenech, A.; Vinyals, O.; Hassabis, D.; Wierstra, D.; and Blundell, C. 2017. Neural episodic control. *arXiv preprint arXiv:1703.01988* .

Pyeatt, L. D. 2003. Reinforcement Learning with Decision Trees. In *Applied Informatics*.

Pyeatt, L. D.; and Howe, A. E. 2001. Decision tree function approximation in reinforcement learning. In *Proc. of the 3rd Int. Symposium on Adaptive Systems: Evolutionary Computation and Probabilistic Graphical Models*.

Quinlan, J. R. 1986. Induction of decision trees. *Machine learning* 1(1).

Rodriguez, I. D. J.; Killian, T. W.; Son, S.; and Gombolay, M. C. 2020. Optimization Methods for Interpretable Differentiable Decision Trees in Reinforcement Learning. In *Proc. of the 23rd Int. Conf. on Artificial Intelligence and Statistics*.

Ross, S.; Gordon, G. J.; and Bagnell, J. A. 2011. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *Proc. of the 14th Int. Conf. on Artificial Intelligence and Statistics*.

Roth, A. M.; Topin, N.; Jamshidi, P.; and Veloso, M. 2019. Conservative Q-Improvement: Reinforcement Learning for an Interpretable Decision-Tree Policy. *arXiv preprint arXiv:1907.01180* .

Rusu, A. A.; Colmenarejo, S. G.; Gulcehre, C.; Desjardins, G.; Kirkpatrick, J.; Pascanu, R.; Mnih, V.; Kavukcuoglu, K.; and Hadsell, R. 2015. Policy distillation. *arXiv preprint arXiv:1511.06295* .

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* .

Sokol, K.; and Flach, P. 2019. Desiderata for Interpretability: Explaining Decision Tree Predictions with Counterfactuals. In *Proc. of the 33rd AAAI Conf. on Artificial Intelligence, Student Abstract Track*.

Sreedharan, S.; Srivastava, S.; and Kambhampati, S. 2020. TLdR: Policy Summarization for Factored SSP Problems Using Temporal Abstractions. In *Proc. of the 30th Int. Conf. on Automated Planning and Scheduling*.

Strehl, A. L.; Diuk, C.; and Littman, M. L. 2007. Efficient structure learning in factored-state MDPs. In *Proc. of the 22nd AAAI Conf. on Artificial Intelligence*.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* 112(1-2).

Tabrez, A.; Agrawal, S.; and Hayes, B. 2019. Explanation-based reward coaching to improve human performance via reinforcement learning. In *Proc. of the 14th Int. Conf. on Human-Robot Interaction*.

Tang, Y.; Nguyen, D.; and Ha, D. 2020. Neuroevolution of Self-Interpretable Agents. *arXiv preprint arXiv:2003.08165* .

Topin, N.; and Veloso, M. 2019. Generation of Policy-Level Explanations for Reinforcement Learning. In *Proc. of the 33rd AAAI Conf. on Artificial Intelligence*.

Tuyls, K.; Maes, S.; and Manderick, B. 2002. Reinforcement learning in large state spaces. In *Robot Soccer World Cup*.

Uther, W. T.; and Veloso, M. M. 1998. Tree based discretization for continuous state space reinforcement learning. In *Proc. of the 15th National Conf. on Artificial Intelligence*.

Vasic, M.; Petrovic, A.; Wang, K.; Nikolic, M.; Singh, R.; and Khurshid, S. 2019. MoËT: Interpretable and Verifiable Reinforcement Learning via Mixture of Expert Trees. *arXiv preprint arXiv:1906.06717* .

Wang, X.; Chen, Y.; Yang, J.; Wu, L.; Wu, Z.; and Xie, X. 2018. A reinforcement learning framework for explainable recommendation. In *Proc. of the Int. Conf. on Data Mining*.

Wang, Z.; Schaul, T.; Hessel, M.; Hasselt, H.; Lanctot, M.; and Freitas, N. 2016. Dueling network architectures for deep reinforcement learning. In *Proc. of the 33rd Int. Conf. on Machine Learning*.

Yang, Z.; Bai, S.; Zhang, L.; and Torr, P. H. 2018. Learn to interpret Atari agents. *arXiv preprint arXiv:1812.11276* .