Conditional Abstraction Trees for Sample-Efficient Reinforcement Learning

Mehdi Dadvar[1] **Rashmeet Kaur Nayyar**[1] **Siddharth Srivastava**[1]

[1]Arizona State University, Tempe, Arizona, USA

Abstract

In many real-world problems, the learning agent needs to learn a problem's abstractions and solution simultaneously. However, most such abstractions need to be designed and refined by hand for different problems and domains of application. This paper presents a novel top-down approach for constructing state abstractions while carrying out reinforcement learning (RL). Starting with state variables and a simulator, it presents a novel domain-independent approach for dynamically computing an abstraction based on the dispersion of temporal difference errors in abstract states as the agent continues acting and learning. Extensive empirical evaluation on multiple domains and problems shows that this approach automatically learns semantically rich abstractions that are finely-tuned to the problem, yield strong sample efficiency, and result in the RL agent significantly outperforming existing approaches.

1 INTRODUCTION

It is well known that *good abstract representations* can play a vital role in improving the scalability and efficiency of reinforcement learning (RL) [Sutton and Barto, 2018, Yu, 2018, Konidaris, 2019]. However, it is not very clear how good abstract representations could be efficiently learned without extensive hand-coding. Several approaches [Kocsis and Szepesvári, 2006, Anand et al., 2015, Jiang et al., 2014] have investigated methods for aggregating concrete states based on similarities in value functions but this approach can be difficult to scale as the number of concrete states or the transition graph grows.

This paper presents a novel approach for top-down construction and refinement of abstractions for sample-efficient reinforcement learning in factored, non-image-based domains. Such problems include several practical applications (e.g., a taxi-management service), where the state is naturally expressed in terms of values of different variables. Translating such states into images would require extensive human effort. Our approach starts with a default, auto-generated coarse abstraction that collapses the domain of each state variable (e.g., the location of each taxi and each passenger in the classic taxi world) to one or two abstract values. This eliminates the need to consider concrete states individually, although this initial abstraction is likely to be too coarse for most practical problems. The overall algorithm proceeds by interleaving the process of refining this abstraction with learning and evaluation of policies, and results in a new form of conditional abstraction that is automatically generated and changes based on the current state to aid learning.

Extensive empirical evaluation on a range of well-established discrete and continuous challenging problems drawn from state-of-the-art RL research [Icarte et al., 2018, Abel et al., 2020, Jin et al., 2022, Barreto et al., 2020] show that *this approach for learning conditional abstractions enables vanilla Q-learning to outperform state-of-the-art baselines* by significantly improving its sample efficiency. In the process, it also learns well-defined abstract representations and draws out similarities across the state space. Furthermore, we found that this approach requires significantly less hyperparameter tuning in comparison to many of the baselines.

Our approach is related to research on variable resolution abstractions for reinforcement learning and abstraction refinement in model checking [Moore, 1991, Clarke et al., 2000, Dams and Grumberg, 2018]. However, unlike existing streams of work, we develop a process that automatically generates semantically rich conditional abstractions, where the final abstraction on the set of values of a variable can depend on the specific values of other variables. For instance, consider a taxi-world problem (Fig. 1). Ideally, when the taxi needs to pick up the orange passenger, a good abstraction would preserve precision in regions closer to the passenger and blur out states where the taxi has a similar

Accepted for the 39th Conference on Uncertainty in Artificial Intelligence (UAI 2023).

policy (Fig. 1 (middle)). However, when the passenger is in the taxi the abstraction should *change* to increase precision around the destination to the extent required to express the taxi's policy for dropping off the passenger (Fig. 1 (right)). In other words, the abstraction on a variable's values (such as the taxi's location) needs to be contingent on the values of the other variables (such as the passenger's presence in the taxi).

To our knowledge, this constitutes the first model-free approach for learning such conditional abstractions on-the-fly while carrying out abstract RL. Our key contributions are (a) formalization and algorithms for building well-defined conditional abstraction trees (CATs) that help compute and represent such abstractions, as well as (b) an algorithm for interleaving RL with CAT learning. While CAT learning could be utilized in numerous RL paradigms, this paper focuses on developing and investigating it for non-image-based domains with discrete actions.

This process also addresses a key challenge in planning with abstractions: it is well known that abstractions of Markovian transition systems such as MDPs are often non-Markovian Singh et al. [1994], Bai et al. [2016], Srivastava et al. [2016]. Intuitively, this is related to the fact that different concrete states represented by an abstract state will have different optimal actions and Q functions. More precisely, the next abstract state depends in general on the agent's current concrete state, whose distribution can depend on the entire action history rather than on only the current abstract state and the current action. To address these problems, CAT+RL carries out RL in the abstract state space but when it observes a high dispersion of temporal difference (TD) errors during Q-learning CAT+RL selectively refines the abstraction, thereby reducing the extent of relevant non-Markov transitions in the abstract state space. In the worst case, this process can lead to a full concretization for discrete state spaces but substantial information is carried over across refinements and the approach turns out to be highly sample efficient in practice. We leave further analysis of this aspect for future work and focus on the core CAT+RL algorithm in this paper.

The presented approach for Conditional Abstraction Trees for RL (CAT+RL) can be thought of as a dynamic abstraction scheme: it provides adjustable degrees of compression [Abel et al., 2016] where the aggressiveness of abstraction can be controlled by tuning the definition of variation in the dispersion of TD errors.

2 RELATED WORK

Abstraction Refinement Several authors have considered variable resolution abstractions and abstraction refinement for RL (e.g., [Moore, 1991, Uther and Veloso, 1998, Whiteson, 2010]). Later work by Seipp and Helmert [2018] developed the concept for classical planning. However, it has remained unclear how to formalize and develop this principle in a manner that provides scalability and sample efficiency in stochastic settings. For instance, Uther and Veloso [1998] employ decision-tree techniques to categorize concrete transition histories rather than creating abstract states. As a consequence, this approach requires a large number of concrete samples for finding a *good* split using multiple sort and search operations on concrete transitions. Whiteson [2010] used the variation in state values as a split metric for tile-based representations of abstract states. This approach requires a deterministic model of the world and needs to keep track of the Q-values of all possible refinements of an abstract state. Additionally, the non-exclusivity of sub-tiles considered during refinement leads to additional computation for sub-tiles that may not be used.

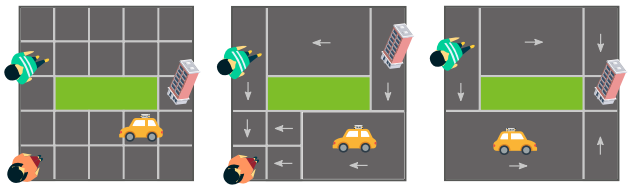

Figure 1: Consider a classic taxi world with two passengers and a building as the drop-off location where the green area is impassable (left). Meaningful conditional abstractions can be constructed, for example, for situations where both passengers are at their pickup locations (middle), or one passenger has already been picked up (right).

The approach presented in this paper addresses these long-standing problems and develops a well-defined formalization that enables dynamic, variable-resolution abstractions for RL. It achieves this by developing the CAT data structure to keep track of heterogeneous abstractions and uses the CAT to define a purely abstract RL process that runs in concert with dispersion-guided abstraction refinement for stochastic settings. CATs enable CAT+RL to identify abstract states with the greatest TD dispersion and provide the useful property that all children of an abstract state in the CAT are mutually exclusive and exhaustive. This makes CAT+RL's RL process more efficient and scalable.

Offline State Abstraction Most early studies focus on action-specific [Dietterich, 1999] and option-specific [Jonsson and Barto, 2000] state abstraction. Further, Givan et al. [2003] introduced the notion of state equivalence to possibly reduce the state space size by which two states can be aggregated into one abstract state if applying a mutual action leads to equivalence states with similar rewards. Ravindran and Barto [2004] relaxed this definition of state equivalence by allowing the actions to be different if there is a valid mapping between them. Offline state abstraction has further been studied for generalization and transfer in RL [Karia and Srivastava, 2022] and planning [Srivastava et al., 2012,

Karia et al., 2022].

Graph-Theoretic State Abstraction Mannor et al. [2004] developed a graph-theoretic state abstraction approach that utilizes the topological similarities of a state transition graph (STG) to aggregate states in an online manner. Mannor's definition of state abstraction follows Givan's notion of equivalence states except they update the partial STG iteratively to find the abstractions. Another comparable method by Chiu and Soo [2010] carries out spectral graph analysis on STG to decompose the graph into multiple sub-graphs. However, most graph-theoretic analyses on STG, such as computing the eigenvectors in Chiu and Soo's work, can become infeasible for problems with large state spaces.

Monte-Carlo Tree Search (MCTS) MCTS approaches offer viable and tractable algorithms for large state-space Markovian decision problems [Kocsis and Szepesvári, 2006]. Jiang et al. [2014] demonstrated that proper abstraction effectively enhances the performance of MCTS algorithms. However, their clustering-based state abstraction approach is limited to the states enumerated by their algorithm within the partially expanded tree, which makes it ineffectual when limited samples are available to the planning/learning agent. Anand et al. [2015] advanced Jiang's method by comprehensively aggregating states and state-action pairs aiming to uncover more symmetries in the domain. Owing to their novel state-action pair abstraction extending Givan and Ravindran's notions of abstractions, Anand et al.'s method results in higher quality policies compared to other approaches based on MCTS. However, their bottom-up abstraction scheme makes their method computationally vulnerable to problems with significantly larger state space size. Moreover, their proposed state abstraction method is limited to the explored states since it applies to the partially expanded tree.

3 BACKGROUND

Markov decision Processes (MDPs) [Bellman, 1957, Puterman, 2014] are defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces respectively. Generally, a concrete state $s \in \mathcal{S}$ can be defined as a set of n state variables such that $\mathcal{V} = \{v_i | i = 1, \ldots, n\}$. In this paper, we focus on problems where the state is defined using a set of variables. $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is a transition probability function, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is a reward function, and γ is the discount factor. A policy π is a solution to an MDP, denoted as $\pi : \mathcal{S} \rightarrow \mathcal{A}$. We consider the RL settings, where an agent needs to interact with an environment that can be modeled as an MDP with unknown $\mathcal{T}$. The objective is to learn an optimal policy that maximizes the long-term cumulative reward for this MDP.

When the size of the space state increases significantly, most RL algorithms fail to solve the given MDP due to the *curse of dimensionality*. Abstraction is a dimension reduction mechanism by which the original problem representation maps to a new reduced problem representation [Giunchiglia and Walsh, 1992]. We adopt the general definition of state abstraction proposed by Li et al. [2006].

Definition 1 *Let $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$ be a ground MDP from which an abstract MDP $\bar{M} = \langle \bar{\mathcal{S}}, \mathcal{A}, \bar{\mathcal{T}}, \bar{\mathcal{R}}, \gamma \rangle$ can be derived via a state abstraction function $\phi : \mathcal{S} \rightarrow \bar{\mathcal{S}}$, where the abstract state mapped to concrete state s is denoted as $\phi(s) \in \bar{\mathcal{S}}$ and $\phi^{-1}(\bar{s})$ is the set of concrete states associated to $\bar{s} \in \bar{\mathcal{S}}$. Further, a weighting function over concrete states is denoted as $w(s)$ with $s \in \mathcal{S}$ s.t. for each $\bar{s} \in \bar{\mathcal{S}}$, $\sum_{s \in \phi^{-1}(\bar{s})} w(s) = 1$, where $w(s) \in [0, 1]$. Accordingly, the abstract transition probability function $\bar{\mathcal{T}}$ and reward function $\bar{\mathcal{R}}$ are defined as follows:*

$$\bar{\mathcal{R}}(\bar{s}, a) = \sum_{s \in \phi^{-1}(\bar{s})} w(s) \mathcal{R}(s, a),$$

$$\bar{\mathcal{T}}(\bar{s}, a, \bar{s}') = \sum_{s \in \phi^{-1}(\bar{s})} \sum_{s' \in \phi^{-1}(\bar{s})} w(s) \mathcal{T}(s, a, s').$$

In this work, we consider a uniform weighting function, i.e., $w(s) = 1$ for all concrete states. When it comes to the decision-making in an abstract MDP, all concrete states associated with an abstract state $\bar{s} \in \bar{\mathcal{S}}$ are perceived identically. Accordingly, the relation between abstract policy $\bar{\pi} : \bar{\mathcal{S}} \rightarrow \mathcal{A}$ and the concrete policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ can be defined as $\pi(s) = \bar{\pi}(\phi(s))$ for all $s \in \mathcal{S}$. Further, the value functions for an abstract MDP are denoted as $V^{\bar{\pi}}(\bar{S})$, $V^*(\bar{S})$, $Q^{\bar{\pi}}(\bar{S}, a)$, and $Q^*(\bar{S}, a)$.

4 OUR APPROACH

4.1 OVERVIEW

Starting with state variables and a simulator, we develop a domain-independent approach for dynamically computing an abstraction based on the dispersion of TD errors in abstract states. The idea of dynamic abstraction is to learn a problem's solution and abstractions simultaneously. We propose a top-down abstraction refinement mechanism by which the learning agent effectively refines an initial coarse abstraction through acting and learning. We illustrate this mechanism with an example.

Example 1 *Consider a 4x4 Wumpus world consisting of a pit at (2,2) and a goal at (4,4). In this domain, every movement has a reward -1. Reaching the goal results in a positive reward of 10 and the agents receive a negative reward -10 for falling into the pit. The goal and the pit are the terminal states of the domain. The agent's actions include moving to non-diagonal adjacent cells at each time step s.t. $\mathcal{A}$ = {up, down, left, right}.*

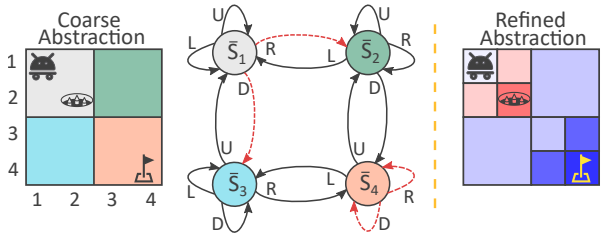

Figure 2: An example of dynamic heterogeneous abstraction refinement for a Wumpus world.

Considering Example 1, Fig. 2 (left) shows a potential initial coarse abstraction in which the domain of each state variable (here x and y coordinates) is split into two abstract values and $\bar{S}_1$ and $\bar{S}_4$ contain the pitfall and goal location respectively. As a result, when learning, the agent will observe a high standard deviation on the TD errors of $(\bar{S}_1, right), (\bar{S}_1, down), (\bar{S}_4, right)$, and $(\bar{S}_4, down)$ because of the presence of terminal states with large negative or positive rewards. Guided by this dispersion of TD errors, the initial coarse abstraction should be refined to resolve the observed variations. Fig. 2 (right) exemplifies an effective abstraction refinement for Example 1 demonstrated as a heatmap of TD errors. Notice that the desired abstraction is a heterogeneous abstraction on the domains of state variable values where the abstraction on a variable depends on the value of the other variables: let x and y domains be $\{1, 2, 3, 4\}$. When $y > 2$, the domain of x (originally $\{1, 2, 3, 4\}$) is abstracted into sets $\{1, 2\}$, $\{3\}$, and $\{4\}$, but when $y \leq 2$, the domain of x is abstracted into sets $\{1\}$, $\{2\}$, and $\{3, 4\}$.

4.2 CONDITIONAL ABSTRACTION TREES

Recall that the value of a state variable v_i inherently falls within a known domain. Partitioning these domains is one possible way to construct state abstractions. The abstraction of one state variable is conditioned on a specific range of any other state variables. Accordingly, we need to maintain and update such conditional abstractions via structures that we call Conditional Abstraction Trees (CATs).

Fig. 3 (right) exemplifies a partially expanded CAT for the problem in Example 1. The tree's root node contains the global ranges (the first range refers to the horizontal location x and the second range refers to the vertical location y) for both of these state variables representing an initial coarse abstraction (in white). The annotations visualize how this initial abstraction can be further refined w.r.t. a state variable resulting in new conditional abstractions (repetitive annotations are not shown for the sake of readability). The refinement procedure of the Wumpus world associated with each level of the tree is also displayed in Fig. 3 (left).

Given the set of state variables $\mathcal{V}$, we define an abstract state

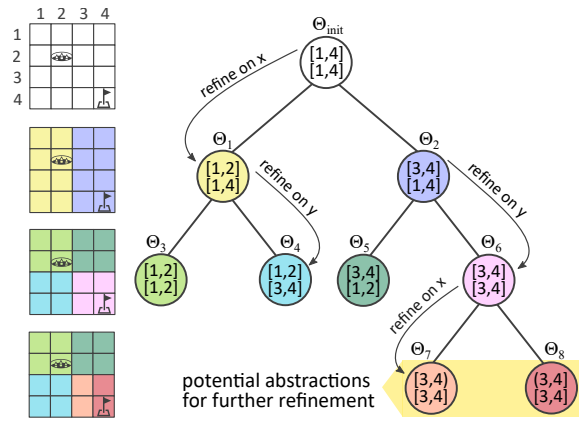

Figure 3: This figure illustrates a Conditional Abstraction Tree (CAT) for Example 1. Ranges written inside the nodes represent $\theta_i \in \Theta$. Each node represents a conditional abstraction.

using the set of partitions, one for each variable v_i, where each partition θ_i is an interval of the form $[l_i, h_i]$. Thus, the coarse abstract state for Example 1 could be defined by $\theta_1 = [1, 4]$ and $\theta_2 = [1, 2]$. An abstraction is defined as $\Theta = \{\theta_i | i = 1, \ldots, n\}$, where $n = |\mathcal{V}|$. In fact, CAT is a hierarchical abstraction tree starting with an initial abstraction Θ_{init} that represents the original range for each state variable $v_i \in s$ s.t. $\Theta_{init} = \{\theta_i | i = 1, \ldots, n$ and $l_i = v_i^{min}$ and $h_i = v_i^{max}\}$, where v_i^{min} and v_i^{max} denote the lower and upper bounds on the range of v_i respectively. In Example 1, there are two state variables so the initial abstraction is $\Theta_{init} = \{[1, 4], [1, 4]\}$. The initial abstraction also induces the starting coarse abstraction since the range for each state variable suggests that all values for all state variables are compressed into one abstract state.

This initial coarse abstraction induced by the initial abstraction Θ_{init} needs to be further refined so that the learning agent can improve its performance through a more fined representation. Let Θ be an abstraction. We define a refinement function $\delta(\Theta, i, f)$ that splits the range of partition $\theta_i \in \Theta$ of state variable v_i into f equal ranges resulting in f new abstractions. Now, we formally define the refinement function $\delta(\Theta, i, f)$.

Definition 2 *Let $\Theta = \langle \theta_1, \ldots, \theta_n \rangle$ be an abstract state for a domain with variables $v_1, \ldots, v_n$. We define the f-split refinement of Θ w.r.t. variable i as $\delta(\Theta, i, f) = \{\Theta^1, \ldots, \Theta^f\}$ where all Θ^j's are the same as Θ on every θ_k for $k \neq i$. $\theta_i = [l, h]$ is partitioned with f new boundaries at least $\|\theta\|/f$ values apart: $l, l_1, l_2, \ldots, l_{f-1}, h$ where $l_x = l + x \times \lfloor (h - l)/f \rfloor$.*

Next, we need to define the relation between two given abstractions in the form of Θ in order to determine if one is obtained by refining the other.

Algorithm 1: State Abstraction

FindAbstract (CAT ξ, Θ_{start}, s):

1: **if** $(\forall v_i \in s)(v_i \in \theta_i^{start})$ **then**
2:   **if** $\Theta_{start} \in L_\xi$ **then**
3:     **return** Θ_{start}
4:   **else**
5:     $children \leftarrow Children(\Theta_{start})$
6:     **for** $\Theta_{child} \in children$ **do**
7:       **if** $(\forall v_i \in s)(v_i \in \theta_i^{child})$ **then**
8:         `FindAbstract` (ξ, Θ_{child}, s)

Definition 3 *Let Ψ be the set containing all possible abstractions. Given $\Theta_a, \Theta_b \in \Psi$, we say Θ_b is obtained by refining Θ_a, denoted as $\Theta_b \triangleright \Theta_a$, iff $(\forall i \in [1,n])(\theta_i^b \subseteq \theta_i^a)$. Moreover, $\Theta_b \triangleright \Theta_a \equiv \Theta_a \triangleleft \Theta_b$. Although this definition determines an ancestral relation between Θ_a and Θ_b, we need to know the factor f by which Θ_a has been refined to determine if Θ_b is the direct result of refining Θ_a. We say Θ_b is obtained directly by refining Θ_a, denoted as $\Theta_b \trianglerighteq \Theta_a$, iff $\exists\ i\ (\theta_i^b \subset \theta_i^a)$, $(\forall k_{\neq i} \in [1,n])(\theta_k^b = \theta_k^a)$ and $|\theta_i^b| \times f = |\theta_i^a|$.*

With these definitions in hand, we can now formally define CAT as a tree to construct and maintain the hierarchy of conditional partitions. A CAT, denoted as ξ, represents a tree structure specifying the hierarchy of conditional abstractions in the form of Θ.

Definition 4 *A conditional abstraction tree (CAT) is defined as $\xi = \{N, E\}$, where N is the set of nodes and E is the set of edges. Each node in N corresponds to an abstraction Θ, s.t. $N = \{\Theta_m | m \in [1, n_\xi]\}$, where n_ξ is the cardinality of CAT and the root node of the tree is the initial abstraction Θ_{init}. Every parent Θ_p and child Θ_c nodes in ξ are connected via an edge e_p^c s.t. $e_p^c \implies \Theta_c \trianglerighteq \Theta_p$. $L_\xi = \{\Theta_m | (\forall k \in [1, n_\xi])(\Theta_k \ntrianglerighteq \Theta_m)\}$ is defined as the set of leaf nodes representing the set of abstract states.*

Given a CAT ξ and a concrete state s, the mapping $\phi(s) : \mathcal{S} \to \bar{\mathcal{S}}$ can be done via a level-order tree search starting from Θ_{init}. The corresponding abstract state $\bar{s}$ is the node Θ_{found} iff $\forall i \in [1,n]\ v_i \in \theta_i^{found}$ (inclusion condition) and Θ_{found} is a leaf node, i.e., $\Theta_{found} \in L_\xi$. Alg. 1 computes the $\phi : \mathcal{S} \to \bar{\mathcal{S}}$ mapping for a given concrete state s under CAT ξ, starting from CAT's root node Θ_{init}. `FindAbstract`(ξ, Θ_{start}, s) starts the level-order search from Θ_{start} and it always finds the corresponding abstract state when $\Theta_{start} = \Theta_{init}$. This algorithm checks the inclusion condition first for Θ_{start} (Line 1 in Alg. 1). If Θ_{Start} is not a leaf node, the algorithm checks the inclusion condition for children of Θ_{start} (Line 7 in Alg. 1) and if a child satisfies the condition, `FindAbstract` gets invoked recursively (Line 8 in Alg. 1).

Any state abstraction under a given CAT ξ induces an abstract representation of the underlying concrete MDP M. Thus, an MDP M can have two abstract representations $\bar{M}_a$ and $\bar{M}_b$ under two CATs ξ_a and ξ_b respectively. We define a relational operation to decide which abstract MDP is finer.

Definition 5 *Given MDPs $\bar{M}_a$ and $\bar{M}_b$ abstracted under ξ_a and ξ_b, we say $\bar{M}_a$ is strictly finer than $\bar{M}_b$, denoted as $\bar{M}_a \succ \bar{M}_b$, iff $\forall \Theta^a \in L_{\xi_a}\ \exists \Theta^b \in L_{\xi_b}\ (\Theta^a \trianglerighteq \Theta^b)$. We also say $\bar{M}_a$ is finer than $\bar{M}_b$, denoted as $\bar{M}_a \succeq \bar{M}_a$, iff $\forall \Theta^a \in L_{\xi_a}\ \exists \Theta^b \in L_{\xi_b}\ (\Theta^a \trianglerighteq \Theta^b \vee \Theta^a = \Theta^b)$.*

4.3 LEARNING DYNAMIC ABSTRACTIONS

Definition 4 formalizes the abstraction tree by which the mapping $\phi(s) : \mathcal{S} \to \bar{\mathcal{S}}$ can be performed using a level-order search (see Alg. 1), while Definition 2 explains how a node of a CAT can be refined w.r.t. a state variable v_i through the refinement function $\delta(\Theta, i, f)$. However, our objective is to interleave RL training with phases of abstraction refinement leading to an enhanced abstract policy $\bar{\pi}$ for a given concrete MDP M. To this end, CAT+RL consists of three phases explained below:

Learning phase. Starting with an initial coarse abstraction, the RL agent interacts with the environment and learns an abstract policy $\bar{\pi}$. The learning phase of CAT+RL is a standard RL routine where the agent learns the abstract policy $\bar{\pi}$ through $\phi(s) : \mathcal{S} \to \bar{\mathcal{S}}$ mapping under a CAT. We employ a vanilla Q-leaning algorithm on the abstract state space as the underlying RL algorithm of CAT+RL.

Abstraction evaluation phase. Since the initial coarse abstraction is likely to be too coarse, CAT+RL should refine the CAT ξ to construct a more effective abstraction. To identify the abstract states that need further refinement, CAT+RL starts the abstraction evaluation phase to collect some samples of TD errors throughout the Q-learning process over abstract states. Thus, in the abstraction evaluation phase, the RL agent continues interacting with the environment via epsilon-greedy variant of the fixed abstract policy $\bar{\pi}$ and CAT+RL evaluates the existing abstraction under the CAT ξ by logging the dispersion of TD errors over abstract states. Let $\beta(M, \xi, \bar{\pi}, n_{eval})$ denote the evaluation function which runs the underlying RL routine for n_{eval} episodes with the fixed stochastic abstract policy for a given MDP M and CAT ξ. Throughout the abstraction evaluation phase, the observed dispersion of TD errors is defined as $\Gamma = \{d_m | m \in [1, n_{visited}]\}$, where $n_{visited}$ is the number of visited abstract states during the abstraction evaluation phase and d_m denotes the set of logged $Q^{\bar{\pi}}(\bar{s}, a)$ values for a visited abstract state $\bar{s}$. When the abstraction evaluation phase is done, β returns the dispersion of TD errors in the form of Γ.

Refinement phase. Once the abstraction evaluation phase is terminated, the dispersion of TD errors Γ will be available from which the refinement phase of CAT+RL can

be initiated. In Γ there might be multiple logs of TD errors for the same pair of abstract state and action $(\bar{s}, \bar{\pi}(a))$. Since the policy was fixed until the agent changes an abstract state throughout the abstraction evaluation phase, a high variation of TD errors of the same pair of $(\bar{s}, \bar{\pi}(a))$ indicates that the abstract state $\bar{s}$ is unstable, i.e., represents significantly disparate concrete states, and requires further refinement. Therefore, the first step of the refinement phase is to find the top k unstable states of the CAT ξ. Let $\texttt{UnstableStates}(\Gamma)$ denote a function that finds the set of unstable states in the form of Θ based on Γ. For each visited abstract state in Γ, `UnstableStates` calculates the maximum normalized standard deviation of TD errors over all actions. Then, `UnstableStates` uses *k-means* clustering technique to find and return the top k unstable states among all of the visited abstract states in Γ. Each unstable state can be refined by splitting into f new states w.r.t a state variable i following the definition of f-split refinement in Definition. 2. However, the question is: what state variable should CAT+RL blame for the observed instability in an unstable state? As discussed, CAT+RL learns an abstract policy $\bar{\pi}$ over abstract states so it maintains and updates the Q-table for abstract states to find the optimal abstract policy. However, for problems with discrete state space, CAT+RL can also maintain and update the Q-table for concrete states. This concrete Q-table can be further used for various applications such as finding contributing state variables for an unstable state. Let $\texttt{UnstableVar}(\Gamma, \Theta)$ denote a function that refines an unstable state, in the form of Θ, w.r.t a state variable that results in the most consistent new abstract states. Basically, splitting an abstract state over a state variable results in f new abstract states. Now, for each newly created abstract state, CAT+RL calculates the normalized standard deviation of the TD errors. Intuitively, if all concrete states under any of the newly created abstract states have TD errors with small standard deviation for the same action a, then splitting over that state variable would be the near-optimal refinement and can potentially decrease/resolve the instability in the abstract state. `UnstableVar` repeats this process for all state variables and chooses the one that minimizes the normalized standard deviation of the underlying TD errors on the concrete level. In Sec. D of the supplementary document, we also presented an alternative approach for `UnstableVar` that aggressively refines an abstract state w.r.t all state variables.

CAT+RL repeats the learning, evaluation, and refinement phases sequentially until the RL agent learns an abstract policy $\bar{\pi}$ and a CAT ξ that successfully and effectively learns the solution and abstractions to the MDP M.

4.4 CAT+RL ALGORITHM

Alg. 2 illustrates the procedure by which the agent learns an MDP's solution and abstractions simultaneously through learning, evaluation, and refinement phases explained in Sec. 4.3. First, the initial coarse abstraction needs to be automatically constructed through initializing Θ_{init}, based on the known ranges for each state variable v_i. Then, a CAT ξ is constructed for Θ_{init} with only the root node (Line 1 in Alg. 2).

Algorithm 2: Learning Dynamic Abstractions

Input: M, f
Output: $\bar{M}, \xi, \bar{\pi}$
1: initialize Θ_{init}, ξ, and $\bar{Q}$
2: **for** $episode = 1, n_{epi}$ **do**
3: $s \leftarrow \texttt{reset()}$
4: **for** $steps$ in $episode$ **do**
5: $\bar{s} \leftarrow \texttt{FindAbstract}(\xi, \Theta_{init}, s)$
6: $a \leftarrow \bar{\pi}(\bar{s})$
7: $s', \bar{r}, done \leftarrow \texttt{step}(\texttt{extend}(a))$
8: $\bar{s}' \leftarrow \texttt{FindAbstract}(\xi, \Theta_{init}, s')$
9: $\bar{\pi} \leftarrow \texttt{train}^{\bar{\pi}}(\bar{s}, \bar{s}', a, \bar{r})$
10: $s, \bar{s} \leftarrow s', \bar{s}'$
11: **if** $\bar{M}$ needs refinement **then**
12: $\Gamma \leftarrow \texttt{evaluate}(M, \xi, \bar{\pi}, n_{eval})$
13: $unstable \leftarrow \texttt{UnstableStates}(\Gamma)$
14: **for** each Θ in $unstable$ **do**
15: $i \leftarrow \texttt{UnstableVar}(\Gamma, \Theta)$
16: $nodes \leftarrow \texttt{refine}(\Theta, i, f)$
17: $\xi \leftarrow \texttt{UpdateTree}(\xi, \Theta, nodes)$
18: **return** $\bar{M}, \xi, \bar{\pi}$

The initial ξ induces an abstract MDP $\bar{M}$ for the given MDP M. Then, the learning phase of CAT+RL starts by employing the Q-learning routine (Lines 2 to 10 in Alg. 2). In this phase, CAT+RL implements a vanilla Q-learning over abstract states and updates Q-values based on samples in the form of $\langle \bar{s}, a, \bar{s}', \bar{r} \rangle$. $\bar{r}$ is computed according to the formulation presented in Definition 1, and $\bar{s}$ and $\bar{s}'$ are returned by the function that we illustrated in Alg. 1. Once the samples are transformed into the form explained above, CAT+RL updates the abstract Q-table in Line 9 of Alg. 2.

Induced by the computed state abstraction, extended actions (taking a concrete action repeatedly until the agent reaches a new abstract state, blockage, or a terminal concrete state) are applied to the environment instead of the concrete actions (Line 7 in Alg. 2). CAT+RL checks the refinement condition (Line 11 in Alg. 2) at the end of each learning episode to initiate an abstraction evaluation phase.

We set CAT+RL to check the recent success rate of the RL agent every n_{check} episodes where the refinement condition evaluates to true if the success rate is below some threshold t_{succ}. The choice of the refinement condition introduces a trade-off. On one hand, we want to obtain a near-optimal abstraction that enables the agent to learn the solution effectively. On the other hand, the abstract policy $\bar{\pi}$ should be trained enough to be used in the abstraction evaluation phase

for refinement purposes. When the refinement condition is true, the algorithm runs the evaluation function β for n_{eval} episodes (Line 12 in Alg. 2). Subsequently, the refinement phase (Lines 13 to 17 in Alg. 2) starts by finding the top k unstable states (Line 13 in Alg. 2). Next, CAT+RL finds the contributing state variable for each unstable state (Line 15 in Alg. 2) and refines it w.r.t. to the contributing state variable (Line 16 in Alg. 2). After refining each unstable state, CAT+RL updates the CAT ξ by adding the new abstract states to the abstraction tree (Line 17 in Alg. 2).

5 EMPIRICAL EVALUATION

To assess the performance of CAT+RL, we implemented the method in Python [1] and evaluated it in five domains. We executed all deep learning experiments for our baselines on two GeForce RTX 3070 GPUs with 8 GB memory running Ubuntu 18.04 and all of our other experiments on 5.0 GHz Intel i9 CPUs with 64 GB RAM running Ubuntu 18.04. We investigated the following questions:

(1) Does CAT+RL improve the sample efficiency of vanilla Q-learning beyond state-of-the-art baselines without any expert knowledge?

(2) Does CAT+RL increase the scalability of its underlying RL algorithm beyond existing methods?

(3) Does CAT+RL learns symmetric structures of tasks?

Selection of test problems For the selection of test problems, we did an extensive literature study to ensure that the chosen problems are drawn from contemporary research and are challenging for state-of-the-art methods. As a result, we conducted empirical analyses on three domains with *discrete states*: Office World adapted from Icarte et al. [2018], Wumpus World derived from Russell and Norvig [2020], Taxi World introduced by Dietterich [2000] and adapted from the OpenAI Gym environment Taxi-v3 [2], and two domains with *continuous states*: Water World based on Karpathy [2015], Icarte et al. [2018] and Mountain Car from Brockman et al. [2016]. We adopted significantly large instances of these domains (except for Mountain Car which has a fixed problem size) compared to the ones used in the previous work in non-imaged-based RL. Besides, all of these domains are *stochastic* problems with varying dimensionality (from 2 to 14). Aside from the main empirical evaluations that are reported in Sec. 5.1, we conducted additional scalability studies (see the supplementary document) on the Office World domain, as a case study, to ensure that the selected test problems challenge the scalability of the state-of-the-art baselines and CAT+RL. The details regarding the domains and task descriptions are included in the supplementary document.

[1] https://github.com/AAIR-lab/CAT-RL.git

[2] https://www.gymlibrary.ml/environments/toy_text/taxi/

Selection of baselines For the comparative study, we selected the following baselines: (1) Option-critic Bacon et al. [2017], (2) JIRP Xu et al. [2020], (3) tabular Q-learning Watkins and Dayan [1992], (4) DQN Mnih et al. [2013], (5) A2C Mnih et al. [2016], and (5) PPO Schulman et al. [2017]. Option-critic is a Hierarchical RL (HRL) approach that discovers options autonomously while learning option policies simultaneously. JIRP, a symbolic state-of-the-art RL method, automatically infers reward machines and policies for RL. We chose these state-of-the-art methods as baselines as they automatically learn different abstract representations such as options and reward machines without requiring any human-engineered inputs. We also chose state-of-the-art deep RL methods: DQN, A2C, and PPO as baselines since multiple layers in their neural network architectures progressively construct state abstractions. We use their implementations from the Stable-Baselines3 [3] framework by Raffin et al. [2019].

Hyperparameters Throughout the empirical evaluations, we ran CAT+RL with $t_{succ} = 0.8$, $n_{eval} = 100$, $n_{check} = 100$, and varying values of k for different domains. One important advantage of CAT+RL over Deep-RL baselines is that CAT+RL has only four parameters, as mentioned earlier, and performs robustly regardless of the value of its parameters as long as they are not set to drastically large or small values within their ranges. On the other hand, we have done extensive hyperparameter tuning for the baselines. The details about the used neural network architectures, parameters, and hyperparameters for baselines and CAT+RL are included in the supplementary document.

We report the mean success rates averaged over the last 100 training episodes along with the standard deviations computed from 10 independent runs for each method and domain. We also report the normalized cumulative reward obtained by evaluating the agent on 10 simulation runs, after stopping training at intervals of 10 episodes. We now discuss our results and analysis in detail below.

5.1 RESULTS

Fig. 4 (top) shows a comparison of success rates achieved by all the methods on all the domains. In Office World, CAT+RL outperforms all the baselines and almost converges to a success rate of 1 in around 2k episodes, whereas, PPO and DQN achieve approximate success rates of 0.8 and 0.65 respectively in around 2.5k episodes and have a high standard deviation. In Wumpus World, CAT+RL converges to a success rate of 1 within 4k episodes and significantly outperforms all the baselines which struggle to learn due to the complexity introduced by pitfalls, obstacles, and size of the environment. In Taxi World, CAT+RL achieves the best performance within 12k episodes of training, while

[3] https://github.com/DLR-RM/stable-baselines3

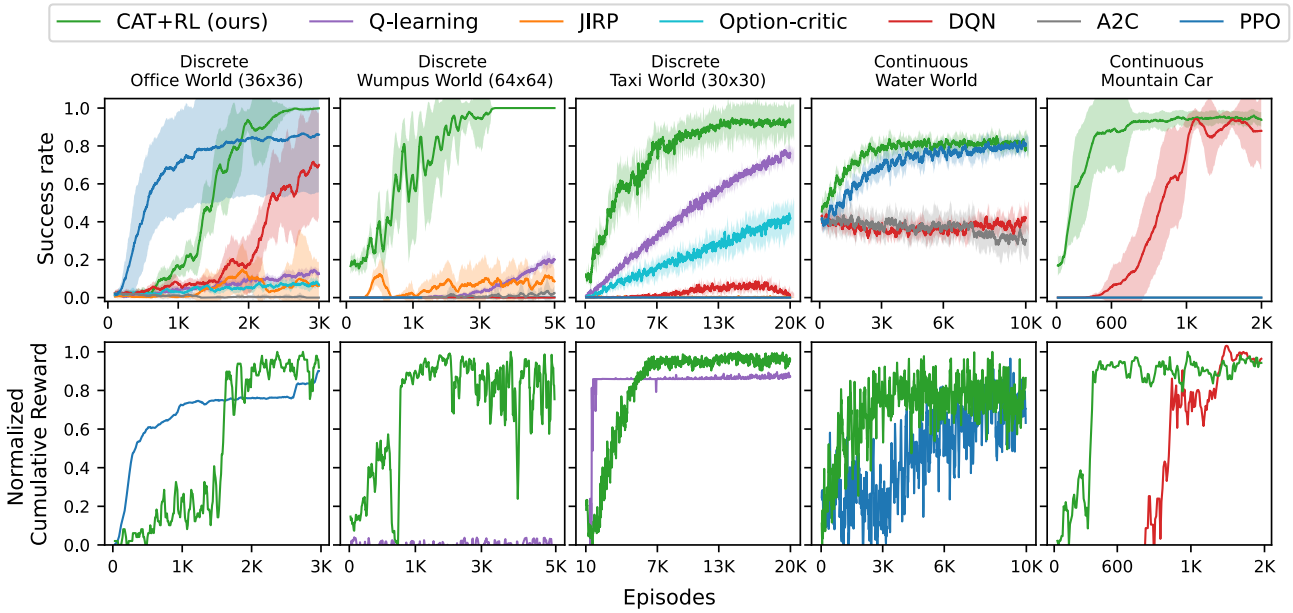

Figure 4: (Top) Success rates (mean and standard deviation) for 10 independent runs averaged over the last 100 training episodes for all the methods, and (Bottom) normalized cumulative reward for 10 simulation runs obtained every 10 training episodes for CAT+RL (ours) and the second-best performing baseline for Office World, Wumpus World, Taxi World, Water World, and Mountain Car. Here, discrete/continuous refers to the state space of the domain.

Q-learning performs better than all other baselines reaching a success rate of 0.75 in 20k episodes. In the Water World domain, CAT+RL learns slightly faster than PPO while all other baselines perform poorly, whereas, CAT+RL learns significantly faster compared to DQN, which is the best baseline, in the Mountain Car domain. We performed further evaluations on CAT+RL and the second-best performing baseline on each domain as shown in Fig. 4 (bottom) by evaluating the policies learned by the agent and comparing the normalized cumulative reward achieved.

Domain	$\lvert\mathcal{S}\rvert$	$\lvert\bar{\mathcal{S}}\rvert$	$\lvert\mathcal{S}\rvert/\lvert\bar{\mathcal{S}}\rvert$
Water World	∞	49144	∞
Mountain Car	∞	13	∞
Taxi World	18000	1552	11.59
Office World	5184	124	41.80
Wumpus World	496	157	26.08

Table 1: Sizes of concrete state spaces and abstract state spaces for the test problems.

Table. 1 draws a comparison between the sizes of the concrete state space and the abstract state space under CAT+RL. As a result of the significant reduction in the size of the concrete state space explained by the abstraction factor in Table. 1, CAT+RL outperforms GPU-based DRL approaches in terms of sample efficiency without relying upon expensive computational hardware and without the corresponding hyperparameter tuning. Additional run-time analyses of CAT+RL and the best-performing baselines are presented in the supplementary document.

5.2 ANALYSIS

We now present our analysis of the three key questions mentioned in Sec. 5.

1. Sample efficiency in the absence of input expert knowledge The results presented in Section 5.1 demonstrate that CAT+RL's sample efficiency is superior to all baselines in both discrete and continuous domains. This is categorically the effect of the learned conditional abstractions by CAT+RL made available to the vanilla Q-learning algorithm. This effect can be perceived from two perspectives: 1) the meaningful conditional abstractions that are automatically constructed by CAT+RL spotlight the most informative aspects of the state space, leading to more sample-efficient learning; and 2) the Q-learning agent benefits from higher levels of exploration over state and action spaces due to the nature of abstraction. This intense exploration can cause more penalization of the agent at the early stages of learning (see cumulative rewards of CAT+RL in Taxi and Office worlds in Fig. 4) but eventually leads to faster learning and superior performance reflected in the success rate.

2. Scalability to larger tasks RL algorithms that learn policy π from a concrete MDP M suffer from the curse of dimensionality as the size of the state space increases. This explains why most of the baselines fail to learn the Wumpus world, as a basic domain, when the size of the problem

increases drastically, as shown in Fig. 4. In contrast, the top-down abstraction refinement scheme of CAT+RL scales effectively to problems with relatively larger state space. As a result, the abstract representations learned by CAT+RL empowered the vanilla Q-Learning algorithm to learn those problems relatively fast and efficiently. We conducted further experiments on scalability and computational complexity of CAT+RL and baselines and the results are presented in the supplementary document.

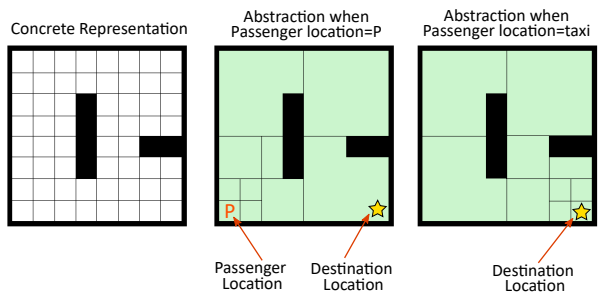

Figure 5: Illustration of two different components of a single CAT learned automatically by CAT+RL for a TaxiWorld problem. Abstraction on "taxi-loc-x" and "taxi-loc-y" changes depending on the value of the passenger-location variable.

3 (a). Abstractions identify sub-tasks within a problem Fig. 5 illustrates the CAT-based abstraction that was computed automatically by CAT+RL for a Taxi domain problem (we are using a small problem instance for clarity in the illustration). When a passenger needs to be picked up, CAT abstraction preserves precision around the passenger's location; when the passenger is in the taxi and needs to be dropped off, cells around the passenger's previous location are no longer significant for distinguishing TD errors, they get merged together, and precision increases around the destination location. This is learned and expressed without human intervention by CAT+RL. It is important to realize that these two abstractions are expressed within one learned CAT—they are different subtrees. One subtree is "active" when the passenger's location is P, and the other is "active" when the passenger's location is the taxi. In an interval abstraction, both parts of the Taxi World (bottom left as well as bottom right) would end up getting refined as episodes continue, thus losing aggregation opportunities and increasing sample complexity. In contrast, CAT abstractions are dynamic (the representation changes depending on the current state) and heterogeneous (the same variable has different "splits" based on the values of other values). This allows our approach to aggregate experience where possible while dynamically increasing resolution on critical-choice paths.

3 (b). Abstractions identify symmetry across sub-tasks One important property of CAT+RL's framework is to construct identical abstractions across the state space for similar sub-problems. This capability of CAT+RL can be useful in large problems where options can be generalized across

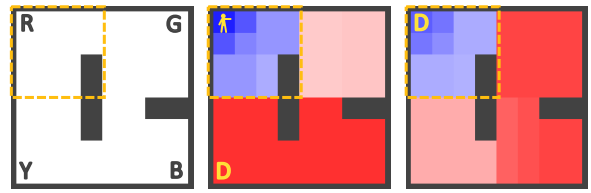

Figure 6: Drawing out similarities across state space of a 8×8 taxi world via CAT+RL's automatic abstraction. Left) R, G, Y, and B are the four predefined pickup/drop-off locations. Middle) Location D is the destination location and the passenger is on the top-left location; and right) Location D is the destination and the passenger is in the taxi.

identically constructed abstractions. Fig. 6 demonstrates two constructed conditional abstractions by CAT+RL for an 8×8 taxi world. In Fig. 6 (middle), the passenger is located at the top-left and the destination is located at the bottom-left of the map. Besides, in Fig. 6 (right), the passenger is in the taxi and the destination is located at the top-left. In both cases, the agent should reach the top-left cell of the map which implies a similarity. CAT+RL discovered this similarity automatically as seen from the generated identical abstractions (highlighted area) for both cases.

6 CONCLUSION

We presented a novel approach for simultaneously learning dynamic abstract representations along with the solution to problems formulated as an MDP. The overall algorithm of CAT+RL proceeds by interleaving the process of refining a coarse initial abstraction with learning and evaluation of policies for the underlying RL agent. We introduced conditional abstraction trees to compute and represent such refined abstractions throughout the CAT+RL procedure. Extensive empirical evaluations demonstrated that CAT+RL effectively enables the vanilla Q-learning algorithm to learn the solution to large discrete and continuous problems, with dynamic representations, where state-of-the-art RL algorithms are outperformed. This superior performance of vanilla Q-learning compared to algorithms with complex neural-network-based architectures is due to CAT+RL's scalable abstraction construction scheme that effectively draws out similarities across the state space and yields powerful sample efficiency in learning. Future work will consider the automatic discovery of generalizable options utilizing the constructed conditional abstract representations by CAT+RL.

Acknowledgements

This work was supported in part by NSF IIS grant 1942856 and ONR grant N00014-23-1-2416.

References

David Abel, David Hershkowitz, and Michael Littman. Near optimal behavior via approximate state abstraction. In *International Conference on Machine Learning*, pages 2915–2923. PMLR, 2016.

David Abel, Nate Umbanhowar, Khimya Khetarpal, Dilip Arumugam, Doina Precup, and Michael Littman. Value preserving state-action abstractions. In *International Conference on Artificial Intelligence and Statistics*, pages 1639–1650. PMLR, 2020.

Ankit Anand, Aditya Grover, Parag Singla, and Mausam. Asap-uct: Abstraction of state-action pairs in uct. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

Aijun Bai, Siddharth Srivastava, and Stuart Russell. Markovian state and action abstractions for mdps via hierarchical mcts. In *IJCAI*, pages 3029–3039, 2016.

André Barreto, Shaobo Hou, Diana Borsa, David Silver, and Doina Precup. Fast reinforcement learning with generalized policy updates. *Proceedings of the National Academy of Sciences*, 117(48):30079–30087, 2020.

Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Chung-Cheng Chiu and Von-Wun Soo. Automatic complexity reduction in reinforcement learning. *Computational Intelligence*, 26(1):1–25, 2010.

Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *International Conference on Computer Aided Verification*, pages 154–169. Springer, 2000.

Dennis Dams and Orna Grumberg. Abstraction and abstraction refinement. In *Handbook of Model Checking*, pages 385–419. Springer, 2018.

Thomas Dietterich. State abstraction in maxq hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, 12, 1999.

Thomas G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of artificial intelligence research*, 13:227–303, 2000.

Fausto Giunchiglia and Toby Walsh. A theory of abstraction. *Artificial intelligence*, 57(2-3):323–389, 1992.

Robert Givan, Thomas Dean, and Matthew Greig. Equivalence notions and model minimization in markov decision processes. *Artificial Intelligence*, 147(1-2):163–223, 2003.

Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*, pages 2107–2116. PMLR, 2018.

Nan Jiang, Satinder Singh, and Richard Lewis. Improving uct planning via approximate homomorphisms. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1289–1296, 2014.

Mu Jin, Zhihao Ma, Kebing Jin, Hankz Hankui Zhuo, Chen Chen, and Chao Yu. Creativity of ai: Automatic symbolic option discovery for facilitating deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 7042–7050, 2022.

Anders Jonsson and Andrew Barto. Automated state abstraction for options using the u-tree algorithm. *Advances in neural information processing systems*, 13, 2000.

Rushang Karia and Siddharth Srivastava. Relational Abstractions for Generalized Reinforcement Learning on Symbolic Problems. *arXiv preprint arXiv:2204.12665*, 2022.

Rushang Karia, Rashmeet Kaur Nayyar, and Siddharth Srivastava. Learning generalized policy automata for relational stochastic shortest path problems. *Advances in Neural Information Processing Systems*, 35:30625–30637, 2022.

Andrej Karpathy. Reinforcejs: Waterworld demo, 2015. *URL http://cs. stanford. edu/people/karpathy/reinforcejs/waterworld. html*, 2015.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

George Konidaris. On the necessity of abstraction. *Current opinion in behavioral sciences*, 29:1–7, 2019.

Lihong Li, Thomas J Walsh, and Michael L Littman. Towards a unified theory of state abstraction for mdps. In *AI&M*, 2006.

Shie Mannor, Ishai Menache, Amit Hoze, and Uri Klein. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the twenty-first international conference on Machine learning*, page 71, 2004.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

Andrew W Moore. Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued state-spaces. In *Machine Learning Proceedings 1991*, pages 333–337. Elsevier, 1991.

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3, 2019.

Balaraman Ravindran and Andrew G Barto. Approximate homomorphisms: A framework for non-exact minimization in markov decision processes. 2004.

Stuart J. Russell and Peter Norvig. Artificial Intelligence: A Modern Approach Fourth Edition, 2020.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Jendrik Seipp and Malte Helmert. Counterexample-guided cartesian abstraction refinement for classical planning. *Journal of Artificial Intelligence Research*, 62:535–577, 2018.

Satinder Singh, Tommi Jaakkola, and Michael Jordan. Reinforcement learning with soft state aggregation. *Advances in neural information processing systems*, 7, 1994.

Siddharth Srivastava, Neil Immerman, and Shlomo Zilberstein. Applicability conditions for plans with loops: Computability results and algorithms. *Artificial Intelligence*, 191:1–19, 2012.

Siddharth Srivastava, Stuart Russell, and Alessandro Pinto. Metaphysics of planning domain descriptions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

William TB Uther and Manuela M Veloso. Tree based discretization for continuous state space reinforcement learning. *Aaai/iaai*, 98:769–774, 1998.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

Shimon Whiteson. Adaptive tile coding. *Adaptive Representations for Reinforcement Learning*, pages 65–76, 2010.

Zhe Xu, Ivan Gavran, Yousef Ahmad, Rupak Majumdar, Daniel Neider, Ufuk Topcu, and Bo Wu. Joint inference of reward machines and policies for reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 590–598, 2020.

Yang Yu. Towards sample efficient reinforcement learning. In *IJCAI*, pages 5739–5743, 2018.