Explainable Reinforcement Learning via Model Transforms

Mira Finkelstein[1], **Lucy Liu**[2], **Nitsan Levy Schlot**[1], **Yoav Kolumbus**[1],
David C. Parkes[2], **Jeffrey S. Rosenshein**[1] and **Sarah Keren**[3]
[1] The Hebrew University of Jerusalem, Benin School of Computer Science and Engineering
[2] Harvard University School of Engineering And Applied Sciences
[3] Technion - Israel Institute of Technology, Taub Faculty of Computer Science

Abstract

Understanding emerging behaviors of reinforcement learning (RL) agents may be difficult since such agents are often trained in complex environments using highly complex decision making procedures. This has given rise to a variety of approaches to *explainability* in RL that aim to reconcile discrepancies that may arise between the behavior of an agent and the behavior that is anticipated by an observer. Most recent approaches have relied either on domain knowledge, that may not always be available, on an analysis of the agent's policy, or on an analysis of specific elements of the underlying environment, typically modeled as a Markov Decision Process (MDP). Our key claim is that even if the underlying MDP is not fully known (e.g., the transition probabilities have not been accurately learned) or is not maintained by the agent (i.e., when using model-free methods), it can nevertheless be exploited to automatically generate explanations. For this purpose, we suggest using formal MDP abstractions and transforms, previously used in the literature for expediting the search for optimal policies, to automatically produce explanations. Since such transforms are typically based on a symbolic representation of the environment, they may represent meaningful explanations for gaps between the anticipated and actual agent behavior. We formally define this problem, suggest a class of transforms that can be used for explaining emergent behaviors, and suggest methods that enable efficient search for an explanation. We demonstrate the approach on a set of standard benchmarks.

1 Introduction

The performance-transparency trade-off is a major limitation of many artificial intelligence (AI) methods [26]: as the inner workings of an agent's decision making procedure increases in complexity, it becomes more powerful, but the agent's decisions become harder to understand. Accordingly, interest in *explainable AI* and the development of transparent, interpretable, AI models has increased rapidly in recent years [5]. This trend is particularly prevalent in *reinforcement learning* (RL), and *deep reinforcement learning* (DRL), where an agent autonomously learns how to operate in its environment. While RL has been successfully applied to solve many challenging tasks, including traffic control [4], robotic motion planning [18], and board games [27], it is increasingly challenging to explain the behavior of RL agents, especially when they do not operate as anticipated. To allow humans to collaborate effectively with RL-based AI systems and increase their usability, it is therefore important to develop automated methods for reasoning about and explaining agent behaviors.

While there has been recent work on explainability of DRL (see [25] for a recent survey), most of these methods either rely on domain knowledge, which may not be available, or on a post-process

Preprint. Under review.

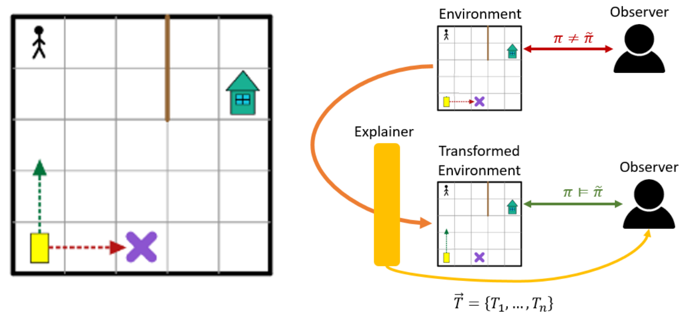

Figure 1: Reinforcement Learning Policy Explanation example (left) and model (right)

analysis of the parameterized policy learned by the agent (e.g., by reasoning about the structure of the underlying neural network [12]). Moreover, most existing methods for explainability do not fully exploit the formal model that is assumed to represent the underlying environment, typically modeled as a Markov Decision Process (MDP) [6], and analyze instead one chosen element of the model (e.g., the agent's reward function [15]).

In this work we focus on RL settings in which the model of the underlying environment is partially known, i.e., the state space and action space are specified via some symbolic representation, but the transition probabilities and reward function are unknown. This is a common assumption in many RL methods, including both *model-based* methods and *model free* methods. In both cases, the action and state spaces are typically given, but the agent is not initially provided with the reward function and transition probabilities and can either learn them explicitly (in model-based RL) or directly learn to optimize its behavior (in model-free RL). For example, in a robotic setting, the agent may have some representation of the state features (e.g., the location of objects) and of the actions it can perform (e.g., picking up an object), but does not know its reward function or the probabilities of action outcomes.

Our key claim is that even if the underlying model may not be fully known, it can nevertheless be used to automatically produce meaningful explanations for the agent's behavior. Specifically, we suggest producing explanations by searching for a set of formal abstractions and transforms that are applied to the MDP used to represent the underlying environment and that yield a behavior that is aligned with an observer's expectations. For this purpose we exploit the rich body of literature that offers a variety of MDP transforms [7, 19, 1] that manipulate different elements of the model by, for example, ignoring the stochastic nature of the environment, ignoring some of the effects of actions, or considering irrelevant constraints [7]. While these methods have so far been used to expedite planning and RL solutions, we use them to automatically produce explanations. That is, while for planning the benefit of using such transforms is in increasing solution efficiency, we use them to isolate features of the environment that cause an agent to deviate from a behavior that is anticipated by an observer.

Formally, we consider an explainability setting, which we term *Reinforcement Learning Policy Explanation* (RLPE), that is comprised of three entities. The first entity, the *actor*, is an RL agent that seeks to maximize its accumulated reward in the environment. The second entity, the *observer*, expects the actor to behave in some way and to follow a certain policy, which may differ from the one actually adopted by the actor. We refer to this policy as the *anticipated policy*, which specifies which actions an observer expects the actor to perform in some set of states [1]. Finally, the *explainer* has access to the description of the environment, to the anticipated policy, and to a set of MDP transforms. The explainer seeks a sequence of transforms to apply to the environment such that the actor's policy in the transformed environment aligns with the observer's anticipated policy.[2]

Example 1 *To demonstrate RLPE, consider Figure 1, which depicts a variation of the Taxi domain [10]. In this setting, the actor represents a taxi that operates in an environment with a single passenger.*

[1] Our formalism can be extended to support cases in which the observer anticipates any one of a set of policies to be realized.

[2] In some settings, the actor and explainer may represent the same entity. We use this structure to separate the role of an actor from the attempt to explain its behavior.

The taxi can move in each of the four cardinal directions, and pick up and drop off the passenger. The taxi incurs a small cost for each action it performs in the environment, and gains a high positive reward for dropping off the passenger at her destination. There are walls in the environment that the taxi cannot move through. The observer has some partial view of the environment and knows which actions the taxi can perform and how it can collect rewards. With the information available to her and the, possibly incorrect, assumptions she makes about the actor's reasoning, the observer anticipates that the taxi will start its behavior by moving towards the passenger (so it can later pick her up and drop her off at her destination). This description of the anticipated behavior over a subset of the reachable states in the environment is the anticipated policy. *The prefix of this policy is depicted by the green arrow in the figure. However, the actual policy adopted by the actor, for which the prefix is represented by the red arrow, is to visit some other location before moving towards the passenger.*

In order to explain the actor's behavior, the explainer applies different action and state transforms to the environment. The objective is to find a transformed model in which the actor follows the anticipated policy. We note that our suggested approach can produce meaningful explanations only if the explainer uses transforms that are meaningful to the observer. In our example, the explainer first applies an action abstraction that allows the taxi to move through walls and trains the actor in the transformed environment. Since the policy in the transformed model still does not match the anticipated policy, the explainer can infer that the reason for the discrepancy is not the fact that the observer may be unaware of the walls in the environment, and therefore this transform would not represent a meaningful explanation. As a second attempt, the explainer applies an abstraction that relaxes the constraint that a car needs enough fuel to be able to move, and allows the taxi to move regardless of its fuel level. After training, the actor's policy in the transformed environment aligns with the anticipated policy. This indicates the observer may not be aware of the fuel constraint, and therefore does not expect the actor to drive towards the gas station before driving to the passenger. This transform can therefore be used to explain the discrepancy between the anticipated and actual policies, and represents an explanation if this constraints can be conveyed to the observer.

Beyond this illustrative example, the ability to understand the "*anticipation gap*" is important in many applications. Examples include autonomous driving, where it is critical to know why a vehicle deviates from an anticipated course of action, medical applications, where it is crucial to explain why an AI system recommends one treatment over another, and search and rescue missions, where a robot is moving in an unknown environment with observations that are different from those of its operator, and may behave in unpredictable ways.

While the translation of the transform sequence that reconciles the gap between the observer and actor to natural language is beyond the scope of this work, since the transforms manipulate the underlying MDP model, they incorporate symbolic information about the environment, and can therefore often be translated to intuitive explanations (e.g., notifying the observer about a missing precondition in its model of some action). Thus, our approach automatically generates explanations without compromising generality. Moreover, while we used a single-agent setting to demonstrate our approach, the same ideas can be applied to collaborative multi-agent settings, where the set of applicable transforms can include, in addition to the transforms used also for single-agent settings, transforms that deal with the multi-agent aspects of the system (e.g., shared resource constraints).

A variety of approaches have been developed recently to provide meaningful post-hoc explanations for the behavior of RL agents. A typical approach in many of these works is to focus on a particular element of the model and investigate its effect on the agent's behavior. For example, in [15] the reward function is decomposed into an aggregation of meaningful reward types, and then actions are classified according to their reward types. In [20], human-designed features, such as the estimated distance to the goal, are used to represent action-value functions. In [14], human-user studies are used to extract saliency maps for RL agents, in order to evaluate the relevance of features with regard to mental models, trust, and user satisfaction, while [12] and [24] use saliency maps to produce visual explanations. Similarly, [3] produce a summary of an agent's behavior by extracting important trajectories from simulated behaviors of the agent.

Our approach supports arbitrary transforms and abstraction that can be applied to the environment model without relying on a particular learning approach of the agent, and can be applied to both single-agent and multi-agent settings. This variety of transforms that can be used for generating explanations relies on the various methods suggested for expediting planning and the search for RL solutions [19]. The work by [30] learns partial symbolic models of the environment and task, and identifies missing preconditions to explain the behavior of an optimal planning agent in deterministic environments. Our framework generalizes to stochastic environments with partially informed RL agents and to arbitrary transforms that can be applied.

The contributions of this work are threefold. First, we present a novel use of formal model transforms and abstractions, formerly mainly used for planning, to produce explanations of RL agent behaviors. Second, we present a formal definition of the Reinforcement Learning Policy Explanation (RLPE) problem and specify classes of state and action abstractions that can be used to produce explanations. Finally, we empirically demonstrate our approach on a set of standard single agent and collaborative multi-agent RL benchmarks.

2 Background

Reinforcement learning (RL) deals with learning policies for sequential decision making of agents that operate in an environment for which the dynamics is not fully known [31]. A common assumption is that the environment can be modelled as a Markov Decision Process (MDP) [6], which is a widely used formalization for sequential decision-making in stochastic environments. An MDP is typically defined as a tuple $\langle S, s_0, A, R, P, \gamma\rangle$, where S is a finite set of states, $s_0 \in S$ is an initial state, A is a finite set of actions, $R : S \times A \times S \to \mathbb{R}$ is a Markovian and stationary reward function that specifies the reward $r(s, a, s')$ that an agent gains from transitioning from state s to s' by the execution of action a, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathbb{P}[\mathcal{S}]$ is a transition function denoting a probability distribution $p(s, a, s')$ over next states s' when action a is executed at state s, and $\gamma \in [0, 1]$ is a discount factor, representing the deprecation of agent rewards over time. In this work we use *factored MDPs* [8], where the set of states is described via a set of random variables $X = X_1, \ldots, X_n$, and where each variable X_i takes on values in some finite domain $Dom(X_i)$. A state is an assignment of a value $X_i \in Dom(X_i)$ for each variable X_i. To model the fully collaborative multi-agent settings we support in this work, we use a *Markov game* [21], which generalizes the MDP model by including *joint actions* $\mathcal{A} = \{A^i\}_{i=1}^n$ representing the collection of action sets A^i, for each of the n agents. We will hereon refer to an MDP as the model of the underlying environment, and highlight the considerations that are specific to a Markov game.

A solution to an RL problem is either a *stochastic policy*, indicated $\pi : S \to \mathbb{P}[A]$, representing a mapping from states $s \in \mathcal{S}$ to a probability of taking an action a at that state, or a *deterministic policy*, indicated $\pi : S \to A$, mapping from states to a single action. The agent's objective is to find a policy that maximizes the expected accumulated reward.

There are a variety of approaches for solving RL problems [32, 31], that can be generally categorized as either *policy gradient methods* (e.g., Stochastic Gradient Ascent) which learn a numerical preference for executing each action, *value-based methods* (e.g., Q-learning), which estimate the values of state-action pairs, and *actor-critic* methods, which combine the value and policy optimization approaches. Another important distinction exists between model-based methods, where a predictive model is learned, and model-free methods, which learn a control policy directly. We support this variety by assuming the algorithm that is used by the actor to compute its policy is part of our input. Our requirement is that each state is characterized by the set of actions that are applicable at that state.

3 MDP Transforms

We use MDP transforms to explain the behaviors of RL agents. Given a large set of possible transforms that is given as input, an explanation is generated by searching for a set of transforms to apply to the environment's model such that the actor's behavior in the modified model aligns with the observer's expectations. Since the transition from the original to the transformed environment is done by manipulating the symbolic representation of the environment, the difference between the models can help the observer reason about the actor's behavior, thus representing an explanation.

We formally define this process in the following sections, and devote this section to describing various transforms suggested in the literature for expediting planning and the search for optimal RL policies, and that we suggest using for explainability. To account for different modifications that can be applied, we define a *transform* as any mapping $T : \mathcal{M} \mapsto \mathcal{M}$ that can be applied to an MDP to produce another MDP. We use the term "transforms" rather than other common terms used in the literature to avoid restricting our discussion to specific kinds of mappings. Thus, transforms may represent “abstractions" or “relaxations", that are typically used to represent mappings that facilitate planning, but also other mappings that may not provide such guarantees, including those that may render more complex environments. Moreover, the transforms we support may modify any element of the MDP. We provide some examples of transforms that can be applied, but our framework supports any transform to produce explanations. We start by defining transforms that modify the representation of the MDP's state space.

Definition 1 (State mapping function) *A* state-mapping function $\phi : S \mapsto S^{\phi}$ maps each state $s \in S$, into a state $s' \in S^{\phi}$. The *inverse image* $\phi^{-1}(s')$ with $s' \in S^{\phi}$, is the set of states that map to s' under mapping function ϕ.

When changing the state space of an MDP, we need to account for the induced change to the other elements of the model. For this, we use a state weighting function that distributes the probabilities and rewards of the original MDP among the states in the transformed MDP.

Definition 2 (State weighting function) *[19] A* state weighting function *of a state mapping function ϕ is function $w_{\phi} : S \mapsto [0, 1]$ where for every $\bar{s} \in S^{\phi}$, $\sum_{s \in \phi^{-1}(\bar{s})} w(s) = 1$.*

Definition 3 (State-Space Transform) *[19] Given a state mapping function ϕ and a state abstraction weighting function w_{ϕ}, a* state space transform $T_{\phi,w}$ *maps an MDP $M = \langle S, s_0, A, R, P, \gamma \rangle$ to $T(M) = \langle \bar{S}, \bar{s_0}, A, \bar{R}, \bar{P}, \gamma \rangle$ where for every action a:*

- $\bar{S} = S^{\phi}$
- $\bar{s_0} = \phi(s_0)$
- $\bar{R}(\bar{s}, a) = \sum_{s \in w^{-1}(\bar{s})} w(s) R(s, a)$
- $\bar{P}(\bar{s}, a, \bar{s}') = \sum_{s \in \phi^{-1}(\bar{s})} \sum_{s' \in \phi^{-1}(\bar{s}')} w(s) P(s, a, s')$

State-space transforms can, for example, group states. In factored representations, this can be easily implemented by ignoring a subset of the state features. In Example 1, a state-space transform can ignore the fuel level, grouping states that share the same taxi and passenger locations.

Another family of transform changes the action space.

Definition 4 (Action mapping function) *An* action mapping function $\psi : A \mapsto A^{\psi}$ *maps every action in A to an action in A^{ψ}. The* inverse image $\psi^{-1}(a')$ *for $a' \in A^{\psi}$, is the set of actions that map to a' under mapping function ψ.*

Various action space transforms have been suggested in the literature for planning with MDPs [23, 2]. Since such transforms inherently bear a symbolic meaning with regards to the environment and agent, a sequence of transforms that yields the anticipated policy represents the gap between the actor's behavior and the expectations of the observer, and therefore can be used as an explanation. We note that the focus here is on automatically finding transforms that bridge this gap, and since they are based on symbolic manipulations of the environment we assume their mapping to natural language is relatively straightforward (but is beyond the scope of this work).

For example, in cases where the transition function of the MDP is assumed to be (partially) specified, it is possible to apply the *single-outcome determinization*, where all outcomes of an action are removed (associated with zero probability) except for one, e.g., the most likely or desired outcome. Similarly, the *all outcome determinization* allows a planner to choose a desired outcome [34]. This transform is typically implemented by creating a separate deterministic action for each possible outcome of the original formulation. If such transforms yield the anticipated policy, it implies that the observer may not be aware of the alternative outcomes of an action or of the stochastic nature of the environment. In settings where actions are associated with preconditions, it is possible to apply a *precondition relaxation*, by which a subset of the preconditions of an action are ignored [30]. For example, for MDPs represented via a factored state space, each action a is associated with a set $pre(a)$ specifying the required value of a subset of its random variables. A precondition relaxation transform removes the restriction regarding these variables. Similarly, it is possible to ignore some of an action's effects, for example by applying a *delete relaxation* and ignoring an actions' effect on Boolean variables that are set to false [13]. A transform may also consider adding preconditions that may considered by the observer by mistake. If such transforms produce the anticipated policy, a plausible explanation is that the observer is not aware of the preconditions or effects of actions, such as in the setting we describe in Example 1.

The transforms mentioned above are applicable to the collaborative multi-agent settings we support in this work. In addition, for such settings we can also apply transforms that are specific to multi-agent settings, such as transforms that allow collisions or that change the communication range in the model. In a multi-agent extension of our taxi example, an observer may not be aware that taxis cannot occupy the same cell - a discrepancy that can be explained by applying a transform that ignores the constraint (precondition) that a cell needs to be empty for an agent to move into it.

4 Transforms as Explanations

We formalize the explainability problem as comprised of three entities: an *actor*, who is an agent operating in the environment, an *observer*, who has some anticipation about the behavior of the actor that may not be met by the actual behavior, and an *explainer*, who wishes to clarify the discrepancy between the anticipated and actual behaviors. The input to a *Reinforcement Learning Policy Explanation* (RLPE) problem includes a description of the environment, a description of the behavior (policy) of an RL agent in the environment, a set of possible behaviors an observer expects the actor to follow and a set of possible transforms that can be applied to the environment.

Definition 5 *A Reinforcement Learning Policy Explanation (RLPE) model is defined as* $R = \langle M, A, \tilde{\pi}, T \rangle$*, where*

- M *is an MDP representing the environment,*
- $A : \mathcal{M} \rightarrow \Pi$ *is the actor, which is associated with an RL algorithm that it uses to compute a policy* $\pi \in \Pi$ *,*
- $\tilde{\pi}$ *is the anticipated policy the observer expects the actor to follow, and*
- $T : \mathcal{M} \rightarrow \mathcal{M}$ *is a finite set of transforms.*

We assume the actor is a reward-maximizing RL agent [3]. The anticipated behavior of the observer describes what the observer expects the actor to do in a subset of its reachable states [4]. Since we do not require the anticipated policy to be defined over all states, we refer to it as a *partial policy* and allow situations in which the observer only specifies its anticipated behavior over selected states. Clearly, the settings of interest here are those in which the actual policy that the actor follows differs from the anticipated policy. We denote by $\mathcal{T}$ the universal set of transforms. Each transform $T \in \mathcal{T}$ is associated with a mapping function for each of the MDP elements that it alters. We let ϕ_T and ψ_T denote the state and action mapping functions, respectively (when the MDP element is not altered by the transform, the mapping represents the identity function). When a sequence of transforms is applied, we refer to the composite state and action mapping that it induces and define it as follows.

Definition 6 (Composite State and Action Space Function) *Given a sequence* $\vec{T} = \langle T_1, \ldots, T_n \rangle$*,* $T_i \in \mathcal{T}$*, the* composite state space function *of* $\vec{T}$*, is* $\phi^{\vec{T}}(s) = \phi_{T_n} \cdot, \ldots, \cdot \phi_{T_1}(s)$*. The* composite action space function *is* $\psi^{\vec{T}}(s) = \psi_{T_n} \cdot, \ldots, \cdot \psi_{T_1}(s)$*.*

The explainer seeks a sequence of transforms that produce an environment where the actor follows a policy that corresponds to the observer's expectations. Formally, we seek a transformed environment where the actor's policy *satisfies* the anticipated policy, i.e., for every state-action pair in the anticipated policy, the corresponding state in the transformed model is mapped to its corresponding action. Given a policy π, we let $\mathbb{S}(\pi)$ represent the set of states for which the policy is defined, and define policy satisfaction as follows.

[3]For the collaborative multi-agent case, instead of a single agent we have a team of agents. All other elements are equivalent.

[4]The model can be straightforwardly extended to support a set of possible anticipated policies

Definition 7 (Policy satisfaction) *Given a partial policy π defined over MDP $M = \langle S, s_0, A, R, P, \gamma \rangle$, a partial policy π' defined over MDP $M' = \langle S', s'_0, A', R', P', \gamma' \rangle$, a state mapping function $\phi : S \mapsto S'$ and an action mapping function $\psi : A \mapsto A'$, π' satisfies π, denoted $\pi' \models \pi$, if for every $s \in \mathbb{S}(\pi)$, $\phi(s) \in \mathbb{S}(\pi')$ and $\psi(\pi(s)) = \pi'(\phi(s))$.*

Clearly, for any two policies there exist state and action mappings that can be applied to cause any policy to satisfy another policy. In order to produce valuable explanations the input needs to includes meaningful transforms to be applied to the underlying MDP, i.e., transforms which change it in a way that highlights the elements of the model that cause unexpected behaviors in the actor's policy. In addition, inspired by the notion of *Minimal Sufficient Explanation* [15], we want to minimize the change that is applied to the environment. Intuitively, the more similar the original and transformed MDPs are, the better the explanation. We therefore assume the input to an RLPE problem includes some distance metric $d : \mathcal{M} \times \mathcal{M} \to \mathbb{R}_+$ between a pair of MDPs, for example by counting the number of atomic changes to the elements of the MDP [29] (see the appendix for a description of some relevant distance metrics from the literature).

The objective of the explainer is to find a sequence of transforms that yield an MDP M' such that the actor's policy in M' satisfies $\tilde{\pi}$. Among the sequences that meet this objective, we are interested in sequences that minimize the distance between the original and the transformed MDP. Formally:

Definition 8 *Given a RLPE model R and a metric function $d : \mathcal{M} \times \mathcal{M} \to \mathbb{R}_+$, an* RLPE problem *seeks a transform sequence $\vec{T} = \langle T_1, \ldots, T_n \rangle$, $T_i \in T$ s.t.*

1. *the actor's policy π' in $\vec{T}(M)$ satisfies $\tilde{\pi}$, i.e, $\pi' \models \tilde{\pi}$,*
2. *among the sequences that satisfy [1], $\vec{T}$ minimizes the distance $d(M, \vec{T}(M))$.*

5 Finding Explanations

In an RLPE setting, the explainer has access to a set of transforms, but does not know a-priori which transform sequence will produce meaningful explanations in a given setting. This means that the explainer may need to consider a large set of possible sequences. Therefore, even though an optimal (i.e., minimum distance) sequence of transforms that satisfies the anticipated policy can be found using an exhaustive search over T, a naive approach is impractical as the number of transform combinations is exponential in $|T|$.

To address this computational challenge, we offer several approaches for expediting the search. Inspired by the search for an optimal MDP redesign in [17], a basic approach is a Dijkstra-like search through the space of transform sequences. Assuming a successor generator is available to provide the MDP that results from applying each transform, the search graph is constructed in the following way. The root node is the original environment. Each edge (and successor node) appends a single transform to the sequence applied to the parent node, where the edge weight represents the distance between the adjacent MDPs according to the distance measure d. For each explored node we examine whether the actor's policy in the transformed MDP satisfies the anticipated policy. The search continues until such a model is found, or until there are no more nodes to explore. The result is a transform sequence that represents an explanation. This approach is depicted in Figure 2, where the top of the figure depicts the search in the transform space, while the lower part depicts the MDPs corresponding to each sequence.

This approach is guaranteed to return an optimal (minimum distance) solution under the assumption that the distance is *additive* and *monotonic* with respect to the transforms in T, in that a transform cannot decrease the distance between the resulting MDP and the original one. From a computational perspective, even though in the worst case this approach covers all the possible sequences, in practice it may find solutions much quicker. In addition, in cases where the transforms are *independent*, in that their order of application does not affect the result, it is possible to expedite the search by maintaining a closed list that avoids the re-computation of examined permutations. The depth of the search can also be bounded by a predefined fixed number of transforms.

In spite of these computational improvements, the above solutions require recomputing an actor's policy in the transformed environment for each explored node. One way to avoid this is by preserving the agent's policy in the original environment and using it for bootstrapping the re-training in

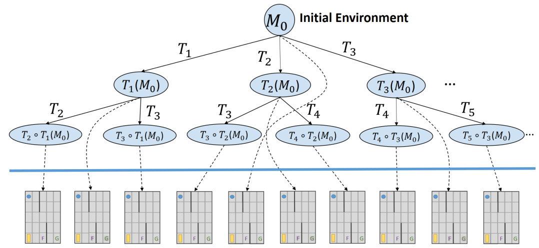

Figure 2: A state-space search in the transform space.

the transformed environment. Another way to expedite the search is to group together a set of transforms and examine whether applying the set leads to a change in the actor's policy. If this compound transform does not change the actor's policy, we avoid computing the values of the specific transforms. This approach is inspired by pattern database (PDB) heuristic approaches [11], as well as the relaxed modification heuristic of [17]. Even though this heuristic approach compromises optimality, it can potentially reduce the computational effort in settings in which aggregation can be done efficiently, such as when transforms have parameterized representations. In our example, if allowing a taxi to move through (all) walls does not change the actor's policy, we avoid computing the value of a single transform of a single wall. Finally, we examine the efficiency of performing a *focused policy update*: when applying a transform, instead of updating the policy for all states, we start by only updating the policy of states that are directly affected by the transform, and then follow the propagated effect of this change. For example, when removing a wall in the taxi domain, we start by updating the policy of states that are near the wall, and iterativly follow the propagated effect of this change to adjacent cells.

6 Empirical Evaluation

Our empirical evaluation is dedicated to examining the ability to produce meaningful explanations via MDP transforms, and to examine the empirical efficiency of our suggested approaches for finding satisfying explanations. Each RLPE setting includes a description of the underlying environment, the actual policy followed by the actor, and the anticipated policy. We describe each component below, before describing our results [5].

Environments: We conducted experiments with 12 different environments, including both deterministic and stochastic domains and single and multi-agent domains (see Figure 3). Among the single agent domains, 3 are adaptations of environments suggested by OpenAI [9]: Frozen Lake , the Taxi domain [10] from Example 1, which we extend with a fuel constraint, and Stochastic Apple Picking, in which agents need to collect apples but have a probability of slipping. We also used seven PDDLGym domains [28]: Sokoban, Blocks World, Towers of Hanoi, Snake, Rearrangement, Triangle Tireworld and Exploding Blocks. The PDDLGym framework aligns with the OpenAI Gym interface while allowing the user to provide a model-based relational representation of the environments. We note that this representation is not available to the actor, which operates using standard RL algorithms without access to the model (e.g., using model-free methods). Our multi-agent domains included extensions of the Sokoban and Taxi domain to collaborative (shared reward) multi-agent settings.

Observer: We consider a partially informed observer which has access to a subset of the environment features. In the taxi domain, for example, the observer may be unaware of the fuel constraint. For all environments we assume the observer anticipates that the actor follows a policy that is optimal w.r.t. the observer's (possibly incomplete or inaccurate) model. Optimal plans are produced using A* implemented using [22].

Actor: We use the DQN, SARSA and CEM from the keras-rl library[6] and Q-LEARNING [33] to represent the actors (note that our framework can be used with any RL algorithm since it is agnostic to the method used by the actor). We trained agents for $600K$-1,000K episodes, with a maximum of 60 steps per episode. Experiments were run on a cluster using 6 CPUs, with 4 cores and 16GB RAM.

[5] our complete dataset, code and results can be

[6] *https://github.com/keras-rl/keras-rll*

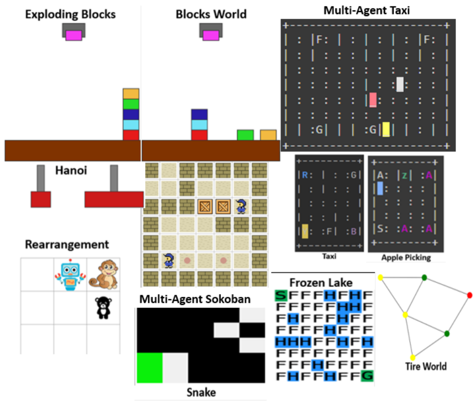

Figure 3: The examined environments

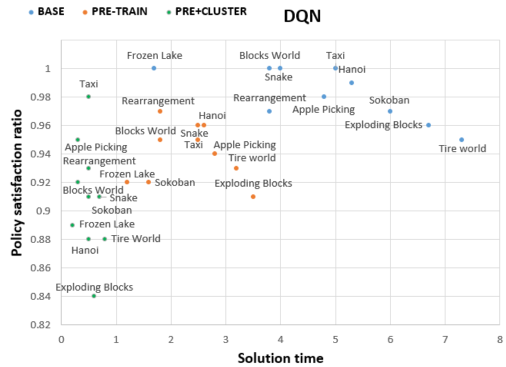

Figure 4: Satisfaction ratio and solution time for DQN

Explainer: We used 5 paramterized transform types: *state space reduction* [34], *likely outcome relaxation* [34], *precondition relaxation* [30], *all outcome determinization* (for stochastic domains) [16], and *delete relaxation* [13]. Grounding is performed automatically for each transform, for the environments in which it is applicable. Each grounded transform modifies a single action or variable.

We used three methods for searching for explanations which were described in the previous section. *BASE* is a Dijkstra search. *PRE-TRAIN* is a Dijkstra search using the learned policy in the original environment to bootstrap the learning in the modified environment, and a focused policy update to avoid iteratively updating the entire policy. *PRE+CLUSTER* extends PRE-TRAIN by computing values of transform sets and using them to prune useless transforms.

Results: To assess the ability to produce explanations using environment transforms, we measured the *satisfaction ratio* of each transform in a given environment. This measure is defined as the fraction of the number of states for which the anticipated policy and actor policy agree among all states for which the anticipated policy is defined, i.e., the number of states $s \in \mathbb{S}(\pi)$ for which $\phi(s) \in \mathbb{S}(\pi')$ and $\psi(\pi(s)) = \pi'(\phi(s))$. We also measured the length of the explanation (i.e. the number of atomic transforms that were applied to the MDP), which we used as our distance measure (and that should be minimized).

Figure 4 presents a scatter plot with the results for the different methods for the single agent domains (due to space limitations, additional results and results for the multi-agent domains can be found in the appendix). It represents for each domain the time it takes (in hours) to find an explanation (x axis) and the satisfaction ratio (y axis) for DQN (the results for the other approaches and measures are in the appendix). Results show that while BASE, achieves the highest satisfaction ratio (which is to be expected from an exhaustive) approach, its computation time is much higher, with up to nine times higher computation time then the best approach for Tireworld. In contrast, PRE+CLUSTER outperforms all other methods in terms of computation time. The approach compromises satisfaction ratio, but with 84% success in the worst case and with an average variance of 0.03 over all environments).

7 Conclusion

We introduced a novel formulation for explainability in RL, and a novel approach for generating explanations using formal model transforms, which were previously primarily used for planning. Our empirical evaluation on a set of single and multi-agent RL benchmarks illustrates the efficiency of our approaches for finding explanations among large sets of transforms.

Possible extensions of our work include integrating human users or models of human reasoning into the process of generating anticipated policies and in the process of evaluating the quality of the explanations generated by our methods. In addition, while this work uses a restrictive satisfaction relation that requires a full match between the anticipated policy and the actor's behavior in discrete domains, it may be useful to use more flexible qualitative evaluation metrics for satisfaction and to account for continuous domains. Finally, our current account of multi-agent settings focuses on fully collaborative settings. We intend to extend our multi-agent account to adversarial domains.

8 Acknowledgments

This research has been partly funded by Israel Science Foundation grant #1340/18.

References

[1] D. Abel, D. Arumugam, L. Lehnert, and M. L. Littman. State abstractions for lifelong reinforcement learning. In *ICML*, 2018.

[2] D. Abel, N. Umbanhowar, K. Khetarpal, D. Arumugam, D. Precup, and M. Littman. Value preserving state-action abstractions. In *AISTATS*, 2020.

[3] D. Amir and O. Amir. Highlights: Summarizing agent behavior to people. In *AAMAS*, 2018.

[4] I. Arel, C. Liu, T. Urbanik, and A. G. Kohls. Reinforcement learning-based multi-agent system for network traffic signal control. *Iet Intelligent Transport Systems*, 2010.

[5] A. B. Arrieta, N. D'iaz-Rodr'iguez, J. D. Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garc'ia, S. Gil-L'opez, D. Molina, R. Benjamins, R. Chatila, and F. Herrera. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion*, 2020.

[6] D. P. Bertsekas. *Dynamic Programming and Optimal Control*, volume I. Athena Scientific, Belmont, MA, USA, 3rd edition, 2005.

[7] B. Bonet and H. Geffner. mgpt: A probabilistic planner based on heuristic search. *JAIR*, 2005.

[8] C. Boutilier, T. L. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *ArXiv*, 1999.

[9] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *ArXiv*, 2016.

[10] T. G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *ArXiv*, 2000.

[11] S. Edelkamp. Automated creation of pattern database search heuristics. In *International Workshop on Model Checking and Artificial Intelligence*. Springer, 2006.

[12] S. Greydanus, A. Koul, J. Dodge, and A. Fern. Visualizing and understanding atari agents. *ArXiv*, 2018.

[13] J. Hoffmann and B. Nebel. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[14] T. Huber, K. Weitz, E. Andr'e, and O. Amir. Local and global explanations of agent behavior: Integrating strategy summaries with saliency maps. *Artif. Intell.*, 2021.

[15] Z. Juozapaitis, A. Koul, A. Fern, M. Erwig, and F. Doshi-Velez. Explainable reinforcement learning via reward decomposition. In *Knowledge-Based Systems*, 2019.

[16] T. Keller and P. Eyerich. A polynomial all outcome determinization for probabilistic planning. *ICAPS*, 2011.

[17] S. Keren, L. Pineda, A. Gal, E. Karpas, and S. Zilberstein. Efficient heuristic search for optimal environment redesign. *ICAPS*, 2019.

[18] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 2013.

[19] L. Li, T. J. Walsh, and M. L. Littman. Towards a unified theory of state abstraction for mdps. In *ISAIM*, 2006.

[20] Z. Lin, K.-H. Lam, and A. Fern. Contrastive explanations for reinforcement learning via embedded self predictions. *ArXiv*, 2021.

[21] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. *Machine Learning Proceedings 1994*, pages 157–163, 1994.

[22] M. Magnaguagno, F. Meneguzzi, and Anwar. pucrs-automated-planning/pddl-parser: Version 1.1, Dec. 2020. URL `https://doi.org/10.5281/zenodo.4391071`.

[23] Mausam and A. Kolobov. Planning with markov decision processes: an ai perspective. *Morgan & Claypool Publishers*, 2012.

[24] T. N. Mundhenk, B. Y. Chen, and G. Friedland. Efficient saliency maps for explainable ai. *ArXiv*, 2019.

[25] E. Puiutta and E. M. Veith. Explainable reinforcement learning: A survey. *ArXiv*, abs/2008.06693, 2020.

[26] C. Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead, 2019.

[27] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. P. Lillicrap, and D. Silver. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 2020.

[28] T. Silver and R. Chitnis. Pddlgym: Gym environments from pddl problems. In *ICAPS-PRL*, 2020. URL `https://github.com/tomsilver/pddlgym`.

[29] J. Song, Y. Gao, H. Wang, and B. An. Measuring the distance between finite markov decision processes. *AAMAS*, 2016.

[30] S. Sreedharan, U. Soni, M. Verma, S. Srivastava, and S. Kambhampati. Bridging the gap: Providing post-hoc symbolic explanations for sequential decision-making problems with black box simulators. *ArXiv*, 2020.

[31] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[32] C. Szepesvari. *Algorithms for Reinforcement Learning*. Morgan and Claypool, 2010.

[33] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Oxford, 1989.

[34] S. W. Yoon, A. Fern, and R. Givan. Ff-replan: A baseline for probabilistic planning. In *ICAPS*, 2007.