Policy Optimization with Sparse Global Contrastive Explanations

Jiayu Yao [1] **Sonali Parbhoo** [1 2] **Weiwei Pan** [1] **Finale Doshi-Velez** [1]

Abstract

We develop a Reinforcement Learning (RL) framework for improving an existing behavior policy via sparse, user-interpretable changes. Our goal is to make minimal changes while gaining as much benefit as possible. We define a minimal change as having a sparse, global contrastive explanation between the original and proposed policy. We improve the current policy with the constraint of keeping that global contrastive explanation short. We demonstrate our framework with a discrete MDP and a continuous 2D navigation domain.

1. Introduction

Human-understandable descriptions of an RL agent's policy are important for human oversight, especially in safety-critical settings (Rudin et al., 2021). Contrastive explanations are known to be effective in helping people understand machine learning models (Jacovi et al., 2021; Rathi, 2019; van der Waa et al., 2018a;b). In the context of RL, contrastive explanations describe the differences in outcomes between the current policy and an alternate one (Puiutta & Veith, 2020). By leveraging the context of the current policy, the outputs of contrastive explanations are often sparser and thus less cognitively demanding than a complete explanation of the new policy (Miller, 2019; Du et al., 2019). When the new policy is very different from the current one, the contrastive explanations can still be dense. Existing works show that in this case, we still want contrastive explanations to be sparse, which are more understandable and executable for humans (Du et al., 2019).

In this work, we build a framework that finds an optimal set of improvements to the current policy (with which the user is already familiar) such that the contrastive explanation given the current and improved policy remains sparse. Our framework consists of two parts: (1) an explanation-

generator that creates global contrastive explanations for two RL policies—to our knowledge, this is the first comprehensive contrastive explanation system for RL—and (2) an optimization procedure for improving the current policy while explicitly constraining the length of the associated explanation (given by the explanation-generator).

Framework Part I: Explanation Generation We consider contrastive explanations of the form:

"Given the current state, doing action A_1 instead of B_1 when you reach state 1, then doing action A_2 instead of B_2 when you reach state 2, ..., then finally doing A_m instead of B_m when you reach state m, will lead to better / worse / unknown differences in outcomes." $\qquad$ (1)

For example, imagine a mobile fitness app that helps users build workout routines. We might explain a new workout plan with "If you mostly do cardio [current state], then after cardio [state 1], doing weightlifting [A_1] instead of stretching [B_1] will improve fat loss [outcome difference]"—this can be more effective than simply describing the new policy, especially if the new policy overlaps significantly with the old policy. While other works have considered contrastive explanations in RL (van der Waa et al., 2018a; Sukkerd et al.), to our knowledge, ours is the first to lay out a *complete* contrast definition that includes the *series* of relevant states and decisions as well as expected changes to future outcomes.

Contribution 1: A method to generate usable contrastive explanations, taking into account scalability, the space of relevant outcomes, and off-policy outcome estimation.

Scalability. Large, continuous domains are common in real life. For example, a fitness app may collect users' demographical information, vital signs, sleeping schedules, etc. This creates two challenges for generating global contrastive explanations: (1) it is computationally expensive to compare policies on all states; (2) it is impractical to explain each and every state to users. *To scale the explanation-generator to large, continuous domains, we develop an approach to perform efficient state exploration and appropriate state abstractions* (Section 4.1 & 4.2).

Relevant outcomes. Current interpretable RL works tend to focus on explaining the optimized outcome (such as the reward function or the Q-value) (Verma et al., 2018; Liu
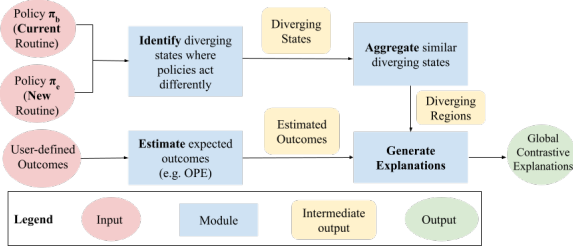
Figure 1: Our explanation-generator takes in two policies, compares two policies in terms of their decisions and expected outcomes, and finally creates a global contrastive explanation by combining the behavioral and the outcome differences.

et al., 2018; Rodriguez et al., 2019; Yau et al., 2020). But in the real world, humans care about multiple outcomes, some of which are often not explicitly considered in the optimization and thus not explained (Doshi-Velez & Kim, 2017; Došilović et al., 2018). For example, an algorithm may aim to increase users' physical activity overall, while users may have more specific goals such as reducing body fat and improving cardiovascular health. *To make the explanations more relevant to users, we allow users to specify a customized set of outcomes* (Section 4.3).

Off-policy outcome estimation. In the setting where only batch data is given, we must be sensitive to two considerations. First, without further interaction with the environment, it is hard to compute the outcomes of the contrasting policy. For example, when proposing a new workout routine, we only have access to the user's past data. Until the new routine is tried out, we will not know the actual fat reduction and cardiovascular health improvement rates. Furthermore, in cases where available data is scarce, it is important to communicate that the model is unsure of the outcomes (Kompa et al., 2021). *In batch settings, we use off-policy evaluation (Hanna et al., 2017) to estimate the outcome of the contrasting policy and report confidence intervals.* (Section 4.3).

Complete Explanation Generator. We build an explanation-generator that includes all the above key features. The complete procedure is illustrated in Figure 1. The generator takes in two policies, either in the batch form or the functional form. It combines the appropriate state abstractions and the estimated outcomes to generate a global contrastive explanation of two given policies (Section 4.4).

Framework Part II: Explanation-Sparsity-Constrained Optimization The explanation-generator in Part I provides us with a way to compactly explain a new policy in terms of changes to the current one. However, if the two policies differ greatly, the explanation in Form 1 will be long. In real life, users often prefer sparse explanations because: (1)

sparsity lessens cognitive demand; (2) users prefer making minimal changes to their current routine while maximizing benefit gain. Thus, when we solve for an optimal new policy, we want to constrain our choices to policies that generate sparse contrastive explanations given our current policy.

Contribution 2: An optimization procedure for improving the current policy that keeps contrastive explanations sparse. We recast policy optimization as the problem of learning how to modify the current policy: we associate a cost to each change to the current policy and find a set of changes that optimizes a single objective, while keeping the total cost within a fixed budget (thus capping the complexity of the associated contrastive explanation). We formalize this optimization as a Constrained Markov Decision Process problem. Note that here, the objective being optimized may or may not be one of the user-defined outcomes in Framework Part I: Explanation Generation.

Empirical Results We demonstrate the efficacy of our framework on two domains: a discrete toy domain, a continuous 2D navigation domain. We show that for explanation generation, our approach generates explanations that are more concise, complete and interpretable comparing to other contrastive explanation forms; for optimization, our optimization converges to policies with sparser contrastive explanations compared to naive policy optimization methods such as policy iteration.

2. Related Work

Interpretable RL. There is a large body of works focusing on learning interpretable policies and explaining the agents' action at a specific decision point (Verma et al., 2018; Liu et al., 2018; Rodriguez et al., 2019; Yau et al., 2020). We consider the task of explaining the difference between *two* policies instead of explaining each policy individually.

We also note that contrastive explanations are different from counterfactual explanations. Counterfactual explanations ask how to alter the past to realize the changes in the outcome, while contrastive explanations ask why one outcome occurs instead of the other (McGill & Klein, 1993).

Contrastive Explanations for non-RL tasks. Several recent works focus on generating contrastive explanations for a specific input (local explanations) in single-step decision-making tasks(van der Waa et al., 2018a; Jacovi et al., 2021). Unlike these methods, we focus on sequential decision-making tasks where *multiple* differences might be encountered in one trajectory. We aim to generate a *global* summary of how *two* policies differ across the entire state space.

Contrastive Explanations for RL tasks. A number of works have explored contrastive explanations for RL. For example, in van der Waa et al., explanations enumerate all

states that appear in the trajectory generated by one policy but not in the trajectory generated by the other. In contrast, we only focus on states where the two policies behave differently, for more concise explanations. Furthermore, van der Waa et al. focus on immediate outcomes while we focus on long-term outcomes. As another example, Sukkerd et al. study a multi-objective setting and help users understand the specific trade-offs between competing objectives of two policies. However, they don't explain *why* objectives are different. In contrast, our explanations are both more complete and compact.

Sparsity in RL Optimization. There is a body of works focusing on generating sparse counterfactual explanations (Wachter et al., 2017; Karimi et al., 2020; Spangher et al., 2018) by either minimizing the distance between the current instance and the counterfactual instance or associating costs to changes of actions. To our knowledge, we are the first to consider performing optimization with respect to the contrastive explanations in an RL setting. That is, we optimize a policy to be minimally different from an existing policy. Similar to prior works, we also associate penalties to changes to the current policy.

3. Notation & Background

Markov Decision Processes. In RL, an agent learns to maximize a reward signal through interactions with the environment (Sutton & Barto, 2018). An RL problem can be formalized as a Markov Decision Process (MDP) – defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space that we assume to be discrete, $\mathcal{T}(s'|s, a)$ is the transition dynamics denoting the probability of reaching state s' by taking action a in state s, and $\mathcal{R}(s, a)$ is the reward function that returns the immediate reward of performing action a in state s. We assume the initial state s_0 is drawn from a distribution $p_0(s)$ and there exists a set of absorbing states $s_T \in \mathcal{S}_T$ with $\mathcal{R}(s_T, \cdot) = 0$. A policy $\pi(a|s)$ defines a distribution over actions given state s.

During optimization, we maximize a single objective – the expected undiscounted return, defined as:

$$\mathcal{J}_\pi = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \mathcal{R}(s_t, a_t) \right]. \tag{2}$$

User Defined Outcomes. Users may also want to know how a policy affects other outcomes. For example, while a clinician may be mainly interested in developing a policy for treating hypotension, they may also be interested in how that policy affects the patient's respiration, mobility, etc.

We denote the user-defined outcomes as a set consisting of M functions of the current state-action pair, $\mathbf{g} = [g^{(0)}(s, a), \dots, g^{(M-1)}(s, a)]^\intercal$ with $g^{(m)}(s_T, \cdot) = 0$. The

expected outcomes for policy π is denoted as $\mathbf{G}_\pi = [G_\pi^{(0)}, \dots, G_\pi^{(M-1)}]^\intercal$ where

$$G_\pi^{(m)} = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} g^{(m)}(s_t, a_t) \right] \quad \forall m \in \{0, \dots, M-1\} \tag{3}$$

We emphasize that the expected outcomes $\mathbf{G}_\pi$ may be independent of the objective $\mathcal{J}_\pi$ that the RL agent is optimizing.

In an online RL setting, both the expected return $\mathcal{J}_\pi$ and the expected outcomes $\mathbf{G}_\pi$ can be computed by rolling out trajectories following the policy π. In a batch setting, where we only have access to a batch of data $\mathcal{D}_b$ collected with the current policy π_b, both terms can be estimated using off-policy evaluation. We denote the estimator of the expected return as $\hat{\mathcal{J}}_\pi$ and the estimator of the expected outcomes as $\hat{\mathbf{G}}_\pi$. We assume that the biases of the estimators $\hat{\mathbf{G}}_\pi, \hat{\mathcal{J}}_\pi$ are sufficiently small so that they do not affect the quality of the contrastive explanations and the optimization performance.

Constrained MDPs. In this work, we study planning under interpretability constraints. That is, we seek optimal changes to our current policy that correspond to sparse contrastive explanations. In Section 5, we will formalize this process as planning under a *constrained MDP* (CMDP).

In an unconstrained MDP setting, the goal is to find a policy π^* that maximizes the expected return $\mathcal{J}_\pi$ in Equation 2:

$$\pi^* = \arg\max_\pi \mathcal{J}_\pi$$

In addition to the standard MDP tuple, a CMDP contains a cost function $\mathcal{C}(s, a)$ with $\mathcal{C}(s_T, \cdot) = 0$ and a threshold κ (in our context, the cost $\mathcal{C}$ will be a penalty for modifying the current policy and the threshold κ will control the sparsity of the corresponding explanation). Now, our goal is to maximize Equation 2 while constraining the expected undiscounted costs:

$$\mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \mathcal{C}(s_t, a_t) \right] \leq \kappa. \tag{4}$$

For a discrete CMDP, the above problem can be written as a finite set of linear inequalities which can be solved using Linear Programming (LP) (Altman, 1999).

The standard CMDP formulation may result in stochastic policies, which are harder to interpret and implement. Thus, we shall use the formulation of Dolgov & Durfee (2005) to obtain deterministic policy solutions to the CMDP, which
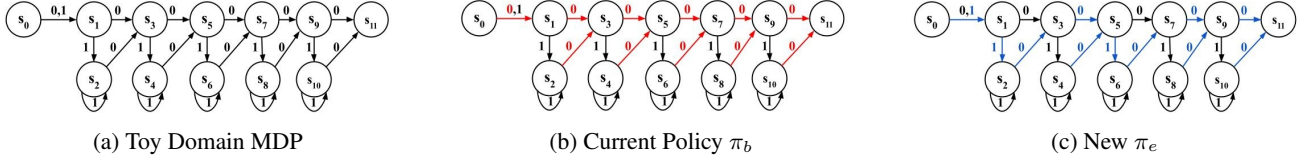
(a) Toy Domain MDP   (b) Current Policy π_b (c) New π_e

Figure 2: Discrete MDP: (a) Definition of the discrete MDP (b) Current Policy π_b with actions highlighted by red arrows (c) New Policy π_e to which we want to compare with actions highlighted by blue arrows.

solves a a mixed integer LP (MILP) as follows:

$$
\begin{aligned}
\arg\max_x \quad & \sum_{s,a} x(s,a)\mathcal{R}(s,a) \\
\text{subject to} \quad & \sum_a x(s',a) - \sum_{a,s} x(s,a)\mathcal{T}(s'|s,a) = p_0(s'), \forall s' \in \mathcal{S} \setminus \mathcal{S}_T \\
& \sum_{a,s} x(s,a)\mathcal{C}(s,a) \leq \kappa \\
& \sum_a \triangle(s,a) \leq 1 \\
& x(s,a)/M \leq \triangle(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A} \\
& x(s,a) \geq 0, \forall s \in \mathcal{S}, a \in \mathcal{A}
\end{aligned}
\tag{5}
$$

In the above, x is the dual variable that can be interpreted as the occupancy measure, which counts the expected discounted number of taking action a in state s. M is a constant to force $0 \leq x(s,a)/M \leq 1$, which can be set to the maximum of the expected return in Equation 2 with $\mathcal{R} = 1$ (i.e. the maximal number of visits to a state). $\triangle_{s,a} \in \{0,1\}$ is a binary variable and the last three inequalities in Equation 5 ensure a unique selection of the action a.

The optimal policy can be calculated by:

$$
\pi(s,a) = \begin{cases} x(s,a)/\sum_s x(s,a) & \text{if } \sum_s x(s,a) > 0 \\ \text{arbitrary} & \text{if } \sum_s x(s,a) = 0 \end{cases}
$$

4. Framework Part I: Explanation Generation

Our first contribution is a complete, global contrastive explanation generator for RL tasks given the current policy π_b and the new policy π_e. In this section, we introduce each module of our explanation-generator with an illustrative toy example, shown in Figure 2 (details in Appendix A.1). Figure 2a depicts the toy MDP while 2b and 2c highlight the actions of the current policy π_b and the new policy π_e, respectively. We set π_b and π_e to be deterministic for pedagogical purposes.

In Section 4.1, we first formalize how to compare two policies in terms of their behaviors. We define a *diverging state*, at which we observe behavioral differences between π_b and π_e. In Section 4.2, we abstract diverging states into human-interpretable *regions* (sets of states) over which the behaviors of π_b and π_e differ. Then, in Section 4.3, we show how to compare policies π_b and π_e in terms of their expected outcomes. Finally, in Section 4.4, we combine the comparison in terms of behaviors and expected outcomes to

generate the global contrastive explanation in Form 1.

4.1. Diverging State Identification

To compare π_b and π_e in terms of behaviors, we first identify states at which the two policies take different actions that lead to different next states. We call such states *diverging states*. Because the policies and the environment can be stochastic, comparing π_b and π_e involves comparing corresponding distributions over actions and next states. In this work, we make a heuristic approximation of the stochastic policies and the environment by focusing on events that occur with high probability. While there are many ways to compare two distributions (e.g. hypothesis testing or divergence measures), we choose this heuristic because policies in real life are frequently nearly deterministic.

Specifically, we denote the most probable action proposed by policies π_b and π_e, in state s, as a_b and a_e, respectively:

$$
a_b = \arg\max_{a \in \mathcal{A}} \pi_b(a|s), \quad a_e = \arg\max_{a \in \mathcal{A}} \pi_e(a|s).
$$

We denote the most probable next state under π_b and π_e as s'_b and s'_e, respectively:

$$
s'_b = \arg\max_{s' \in \mathcal{S}} \mathcal{T}(s'|s, \pi_b), \quad s'_e = \arg\max_{s' \in \mathcal{S}} \mathcal{T}(s'|s, \pi_e).
$$

Given any state s, we define a function $h_{\text{id}} : \mathcal{S} \rightarrow \{0,1\}$ mapping a state s to a label 1 (i.e. s *is* a diverging state) if it satisfies *both* of the following conditions:

$$
\begin{cases}
a_b \neq a_e \text{ or } |\pi_b(a_b|s) - \pi_e(a_e|s)| > \kappa_\pi \\
s'_b \neq s'_e \text{ or } |\mathcal{T}(s_b|s, \pi_b) - \mathcal{T}(s_e|s, \pi_e)| > \kappa_\mathcal{T}
\end{cases}
\tag{6}
$$

where $\kappa_\pi, \kappa_\mathcal{T}$ are domain-dependent constants. The first condition states that the policies π_b and π_e are different when they take different actions in state s with high probability. That is, either: (a) the most probable actions under π_b and π_e are different, or (b) the most probable actions are identical, but with very different associated probabilities. The second condition ensures that even if the policies take different actions with high probability, the agent should not end in the identical next state with high probability. For example, in the toy example, state s_1, s_5 are diverging states while s_0 is not a diverging state, as s_0 fails to satisfy the second condition (others states fail to satisfy the first one).

Application to Continuous Domains. In a large or continuous domain, computing $h_{id}(s)$ for a representative number of states can be computationally expensive. Even if we use sampling, collecting enough samples to cover the state space $\mathcal{S}$ to approximate the function $h_{id}(s)$ may be prohibitive.

We get around this issue by computing h_{id} only for states that are probable under our policies. To do this, we generate trajectories using the recursive Algorithm 1, in which whenever encountering a divergence state, we split the current trajectory into two by rolling out π_b and π_e separately. By rolling out trajectories switching between policies, the algorithm ignores states that cannot be reached by either policy. Note that we split the trajectory whenever we see policies are different, since we expect earlier differences in the trajectory to have more effect on the expected outcomes.

Algorithm 1 includes a hyper-parameter $d_{\max}$ that controls the number of times we are allowed to split. This allows the user to trade-off between exploration of the state space at an early state and the computational complexity of the roll-out (which grows exponentially with the number of splits). In practice, we find that a small $d_{\max}$ often covers a sufficient amount of diverging states for later tasks in our framework.

Algorithm 1 Diverging_State_Collection $(\pi_1, \pi_2, s, d, d_{\max})$

1: Initialize $\mathcal{D} = (\mathcal{S}, \mathcal{Y}), \mathcal{I} = \{\}, k = 1$
2: **if** $d \geq d_{\max}$ **then**
3:     return $\mathcal{D}$
4: **end if**
5: **while** $s \notin \mathcal{S}_T$ **do**
6:     Observe the current state s
7:     $a_1 = \arg\max \pi_1(a|s)$, $a_2 = \arg\max \pi_2(a|s)$
8:     **if** $h_{id}(s, \pi_1, \pi_2) = 1$ **then**
9:         $\mathcal{D}' = $ Diverging_State_Collection $(\pi_1, \pi_2, s, d+1, d_{\max})$
10:         $\mathcal{D}' \rightarrow \mathcal{D}$
11:         **if** key $(a_1, a_2) \notin \mathcal{I}$ **then**
12:             add key: $\mathcal{I}[(a_1, a_2)] = k, k = k+1$
13:         **end if**
14:         $(s, \mathcal{I}[(a_1, a_2)]) \rightarrow \mathcal{D}$
15:     **else**
16:         $(s, 0) \rightarrow \mathcal{D}$
17:     **end if**
18:     Perform action a_1
19: **end while**
20: return $\mathcal{D}$

In the batch setting, we identify the diverging states by calculating $h_{id}(s)$ for every state in the batch data using a similar algorithm (Appendix B.1).

4.2. Diverging State Aggregation

After labeling the state space (Section 4.1), we may have identified a large number of diverging states. As such, a complete enumeration of the diverging states is unlikely to be useful to end-users. Thus, to summarize the difference between diverging and non-diverging states, we aggregate the diverging states into *diverging regions* and extract general descriptions of these regions.

To perform state aggregation, we train an interpretable, rule-based classifier to predict the behavioral difference between π_b and π_e given a diverging state s. The split rules of this classifier correspond to our diverging regions.

Specifically, Algorithm 1 will return a dataset with N samples $\{(s_n, y_n)\}_{n=1}^N$, where $s_n \in \mathcal{S}$ and labels $y_n \in \{0, \ldots, K\}$. We set $y_n = 0$ if s_n is not a diverging state and $y_n = k$ otherwise, with each $k \in \{1, \ldots, K\}$ representing a unique action pair (a_b, a_e). Let the state space $\mathcal{S}$ be partitioned into $\bigcup_{k=0}^K \mathcal{S}_k$ where $\mathcal{S}_k$ contains the samples with labels $y_n = k$. We learn an interpretable classifier $h_{aggr} : \mathcal{S} \rightarrow \mathcal{Y}$ that takes the states s_n as input and predicts the label y_n. From the classifier h_{aggr}, we extract rules for describing each partition $\mathcal{S}_k$.

We have many choices of interpretable classifiers to partition the space of diverging regions. In this work, we adapt Boolean Decision Rules via Column Generation (BDCG) (Dash et al., 2018) to generate rule-based explanations for diverging regions (Appendix B.2), as this is a simple interpretable model with desirable regularization properties. Other interpretable classifiers may involve developing self-explanatory models (Quinlan, 1987; Zhang et al., 2018; Sabour et al., 2017) or provide other forms of global insights for pre-trained models (Altmann et al., 2010; Karpathy et al., 2015). We encourage the developers to use any interpretable classifier appropriate for their tasks.

4.3. Expected outcome estimation

In the previous section, we describe how policies π_b and π_e differ in terms of actions. In this section, we quantify the difference between policies in terms of the expected outcomes for a set of user defined functions (Equation 3).

In practice, we usually have enough data for a good estimate of the expected outcomes of π_b, $\hat{\mathbf{G}}_{\pi_b}$: we either have access to the environment $\mathcal{T}$ in the online setting or enough data collected from π_b in the batch setting. But we may not have enough information to obtain a good estimate of the expected outcomes of π_e, $\hat{\mathbf{G}}_{\pi_e}$. In these situations, it is important to inform the end-users of our uncertainty over the estimated expected outcomes in addition to the estimates themselves. Thus, when comparing the expected outcomes $\hat{\mathbf{G}}_{\pi_b}$ and $\hat{\mathbf{G}}_{\pi_e}$, we incorporate the confidence interval $[l(\mathbf{G}_{\pi_e}), u(\mathbf{G}_{\pi_e})]$ into our estimate $\hat{\mathbf{G}}_{\pi_e}$.

In particular, given an initial state s_0, we compute the difference in expected outcomes $\mathbf{h}_{oc} = [h_{oc}^{(0)}), \ldots, h_{oc}^{(M-1)})]^\intercal$ as:

$$h_{oc}^{(m)}) = \begin{cases} 1 & \text{if } \hat{G}_{\pi_b}^{(m)} < l(G_{\pi_e}^{(m)}) (\pi_e \text{ is better}) \\ -1 & \text{if } \hat{G}_{\pi_b}^{(m)} > u(G_{\pi_b}^{(m)}) (\pi_b \text{ is better}) \\ 0 & \text{otherwise (unknown difference)} \end{cases}$$

Given the transition dynamics $\mathcal{T}$, the expected outcomes es-
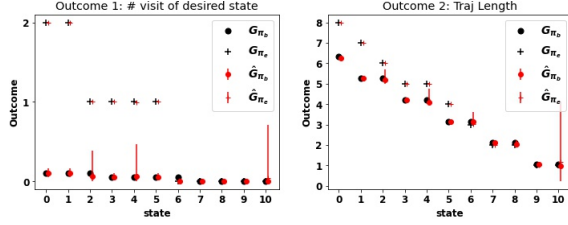
Figure 3: The true outcomes $\mathbf{G}_{\pi_b}$, $\mathbf{G}_{\pi_e}$ and the estimated outcomes $\hat{\mathbf{G}}_{\pi_b}$, $\hat{\mathbf{G}}_{\pi_e}$ of discrete MDP: Black markers represent true outcomes while red ones represent estimates. The estimates are accurate as black and red markers overlap.

timates $\hat{\mathbf{G}}_{\pi_b}$ and the confidence intervals $[l(\mathbf{G}_{\pi_e}), u(\mathbf{G}_{\pi_e})]$ can be obtained using Monte Carlo roll-outs. In the batch setting, those terms can be estimated using off policy evaluation methods such as importance sampling (Thomas & Brunskill, 2016) or fitted-Q evaluation (Hao et al., 2021). We choose model-based Bootstrapping (Hanna et al., 2017), since for our simple domains, we can assume enough data to learn a good transition dynamics estimate, $\hat{\mathcal{T}}$.

For example, in the toy domain (Figure 2), we are interested in: (1) the number of times we visit the desired states (state s_2, s_6) and (2) the trajectory length. From Figure 3, we see that given the initial states $s_0 \cdots s_5$, policy π_e visits the desired states more times (while yielding a longer trajectory) than policy π_b. But given the initial states $s_6 \cdots s_{10}$, we are unsure of the differences in outcomes of the two policies.

4.4. Global Contrastive Explanation Generation

Thus far, we have h_{aggr} that aggregates the diverging states into human-interpretable regions, and $\mathbf{h}_{\text{oc}}$ that computes the differences between the expected outcomes of policies π_b and π_e. We are now ready to generate a global contrastive explanation for π_b and π_e in the format of Form 1, summarizing the behavioral and outcome differences between the two.

The detailed procedure is as follows (Algorithm 2):

1. Given an initial state s_0, we perform a roll-out following π_e. Here, we can use the true environment $\mathcal{T}$, or the estimate $\hat{\mathcal{T}}$. We track the set of diverging regions visited in our roll-out and define the trajectory of visited regions as the *diverging path*: $\mathcal{P}(s_0) = \{s_{k_1}, s_{k_2}, \cdots\}$, where $s_{k_i} \in \mathcal{S}_k$ is defined in Section 4.2.

2. We compute the difference in the expected outcomes of π_b and π_e using h_{oc} from Section 4.3.

3. Different initial states may result in different diverging paths and outcome differences. To make the explanation more compact, we further aggregate the initial states by training an interpretable classifier (the adapted

BDCG) that maps the initial state s_0 to the tuple of the diverging path $\mathcal{P}(s_0)$ and outcome difference $h_{\text{oc}}(s_0)$.

4. We extract the interpretable rules of the classifier from Step 3, which describes the relationship between an initial state and the resulting combination of the diverging path and outcome differences.

Algorithm 2 Generate_Explanations(h_{aggr}, $\mathbf{h}_{\text{oc}}$, π_b, π_e)

1: Initialize $\mathcal{D} = (\mathcal{S}, \mathcal{Y})$, $\mathcal{I} = \{\}$, $z = 0$, episode length L
2: **for** $s_0 \sim p_0$ **do**
3:    Initialize $\mathcal{P}(s_0) = []$
4:    **while** $s \notin \mathcal{T}_T$ **do**
5:       Observe the current state s
6:       **if** $h_{\text{aggr}}(s) \neq 0$ **then**
7:          $s \to \mathcal{P}(s_0)$
8:       **end if**
9:       Perform $a \sim \pi_e$
10:   **end while**
11:   **if** key $(\mathcal{P}(s_0), \mathbf{h}_{\text{oc}}(s_0)) \notin \mathcal{I}$ **then**
12:       add key: $\mathcal{I}[(\mathcal{P}(s_0), \mathbf{h}_{\text{oc}}(s_0))] = z, z = z + 1$
13:   **end if**
14:   $(s_0, \mathcal{I}[(\mathcal{P}(s_0), \mathbf{h}_{\text{oc}}(s_0))]) \to \mathcal{D}$
15: **end for**
16: Train a classifier $h_{\text{exp}} : \mathcal{S} \to \mathcal{Y}$

For the toy domain (Figure 2), we identify states s_1, s_5 as the diverging states (also as diverging regions because the MDP is discrete). We also compute the difference of the expected outcomes in Section 4.3. Together, this yields the following global contrastive explanation:

```
Starting from initial region s_0, in region
s_1, doing action 1 instead of action 0 and
then in region s_5, doing action 1 instead
of action 0 will lead to longer trajectory
but more visits to desired states;

Starting from initial region s_1, in region
s_1, doing action 1 instead of action 0 and
then in region s_5, doing action 1 instead
of action 0 will lead to longer trajectory
but more visits to desired states;

Starting from initial region s_2, in region
s_5, doing action 1 instead of action 0 will
lead to longer trajectory and more number
of visits to desired states;

Starting from initial region s_3, in region
s_5, doing action 1 instead of action 0 will
lead to longer trajectory and more number
of visits to desired states;

Starting from initial region s_4, in region
s_5, doing action 1 instead of action 0 will
lead to longer trajectory and more number
of visits to desired states;

Starting from initial region s_5, in region
s_5, doing action 1 instead of action 0 will
lead to longer trajectory and more number
of visits to desired states;
```

```
Starting from initial region s₆···s₁₀, two
policies, πb and πe act the same.
```

5. Framework Part II: Policy Optimization

In Framework Part I, we describe a generator of complete, contrastive explanations in the RL setting. Now, we use these explanations to define interpretability constraints for policy optimization. That is, we illustrate how to modify the current policy π_b such that (a) the improved policy π_e increases expected returns (Equation 2) and (b) the contrastive explanation (as defined in Framework Part I) given π_b and π_e will be sparse. We formalize this goal as a CMDP that maximizes returns subject to a constraint on the number of changes made to the current policy π_b (which corresponds to a sparse contrastive explanation).

5.1. CMDPs for Sparse Changes in Discrete Domains

In Section 3, we introduced background information on CMDPs. Here, we recast our policy optimization problem over discrete domains as a CMDP. Since we want to encourage sparse modifications, we define the constraint as

$$\mathcal{C}(s, a) = \mathbb{1}_{\{a \neq [\arg\max_a \pi_b(a|s)]\}}, \quad \mathcal{C}(s_T, \cdot) = 0.$$

That is, every modification to the current policy π_b occurred during a roll-out incurs a constant cost and we bound the expected number of changes to the current policy π_b. In addition, we assume the new policy is deterministic as it is more interpretable and executable. We can control the length of the contrastive explanation by varying the threshold κ (smaller κ implies a more similar policy to π_b and a sparser explanation, and vice versa).

5.2. Extensions to Continuous Domains

For continuous domains, we cannot rewrite the optimization using the LP formulation in Equation 5, because now we have an infinite number of dual variables $x(s, a)$.

In this section, we show that, if we have a set of candidate improved policies (produced by RL algorithms or provided by experts), we can use this information to transform a continuous-state MDP into a discrete one to which we can apply the CMDP framework from Section 3. Assuming a set of suggested policies is reasonable in our setting, as we can run RL algorithms to identify possible new policies that improves upon our current one. However, in this case, we still want to choose a new policy π_e that maximizes expected returns with minimal changes to π_b.

To find the optimal π_e, we only need to study regions where candidate and current policies disagree, that is, the diverging regions defined in Section 4.2. Thus, we can recast the optimization problem as one over the discrete state space consisting of diverging regions.

Denote the set of candidate policies as $\{\pi_{e_j}\}_{j=1}^J$. For each candidate policy π_{e_j}, we compare π_{e_j} to the current policy π_b and obtain a set of diverging regions following Section 4.1 and 4.2. We then collect all diverging regions, $\{\mathcal{S}_1, \mathcal{S}_2, \cdots, \mathcal{S}_K\}$, and denote the set containing the absorbing state region as well as the diverging regions as $\mathcal{S}' = \mathcal{S}_T \bigcup (\bigcup_{k=1}^K \mathcal{S}_k)$. We set $\mathcal{S}'$ as our new state space.

By construction, our new state space $\mathcal{S}'$ is finite (ruling out severe pathology in π_{e_j}) and hence discrete. The action space remains the same. We define the transition dynamics $\mathcal{T}'(s_{k'}|s_k, a)$ on the new discrete state space as the probability of transitioning from the region s_k to the region $s_{k'}$ by taking action a. We define a new reward function $\mathcal{R}'(s_k, a)$ by taking the average reward resulting from taking action a in the region s_k. Details of the construction and estimation of the new discrete CMDP over diverging regions can be found in Appendix C. Finally, we can recast the problem of finding an optimal policy π_e as solving the CMDP (Equation 5) defined by our discretized MDP – that is, we find π_e that differs from the currently policy π_b over a minimal set of diverging regions, and that attains the highest gain in expected return (averaged over these regions).

6. Experiments & Results

Taken as a whole, our framework (explanation generation and policy optimization) improves the current policy with sparse and user-interpretable changes. We demonstrate our framework on a toy domain (the discrete MDP in Figure 2) and a continuous 2D navigation domain.

6.1. 2D Navigation Domain

We present a 2D domain (Figure 5) where the agent starts from the initial region and aims to navigate to a goal region. The trajectories of the current policy π_b are given in Figure 8a. We are also given two proposed policies π_{e_1} (Figure 8b) ,π_{e_2} (Figure 8c). As users, we want to stay in the desired region ($0.2 \leq y < 0.3$) while collecting as much reward as possible (Details of the domain in Section A.2).

6.2. Explanation Generation

We first show that our explanation-generator creates a compact explanation given two policies. For the discrete MDP, an example of the explanation is given in Section 4.4. For the 2D domain, we generate a contrastive explanation given the current policy π_b and the optimal policy π^* (Figure 5a) without any sparsity constraints:

"Starting from the initial region, going north instead of east when reaching region $0 \leq x \leq 0.1, 0.2 \leq y \leq 0.3$, then going east instead of south when reaching region $0.1 \leq x \leq 0.2, 0.3 \leq y \leq 0.4$, then going east instead of south when reaching region $0.4 \leq x \leq 0.5, 0.3 \leq y \leq 0.4$, will result in less stay in the desired region ($0.2 \leq y < 0.3$) but more collected rewards.
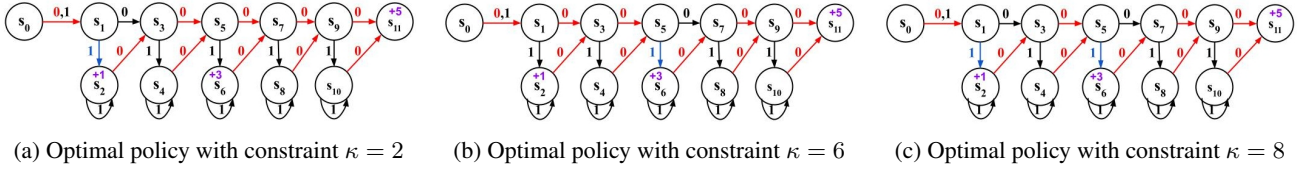
(a) Optimal policy with constraint $\kappa = 2$ (b) Optimal policy with constraint $\kappa = 6$ (c) Optimal policy with constraint $\kappa = 8$

Figure 4: Discrete MDP: Visualization of optimal deterministic policies when increasing κ. The purple number denotes the immediate rewards. Red arrows highlight the current policy behavior while the blue ones represent the changed behavior. The number of changes is aggregated over all initial states.
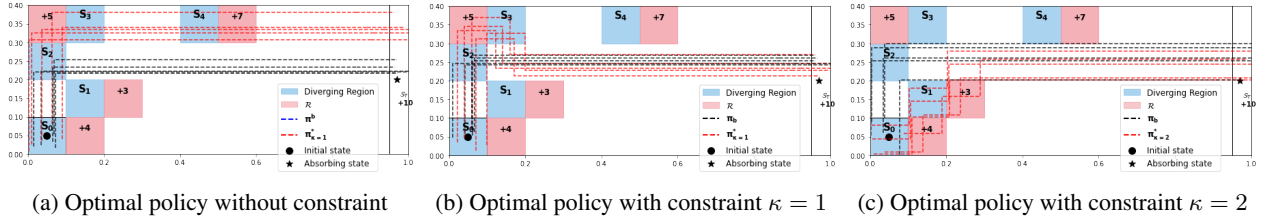

(a) Optimal policy without constraint   (b) Optimal policy with constraint $\kappa = 1$ (c) Optimal policy with constraint $\kappa = 2$

Figure 5: 2D Navigation: Visualization of optimal policy trajectories when increasing κ. The rectangle with the black circle represents the initial state region while the one with the star represents the absorbing state region. Each blue box represents one diverging region while each red box represents regions with different rewards. Each step has a cost of -0.001.

6.3. Policy Optimization

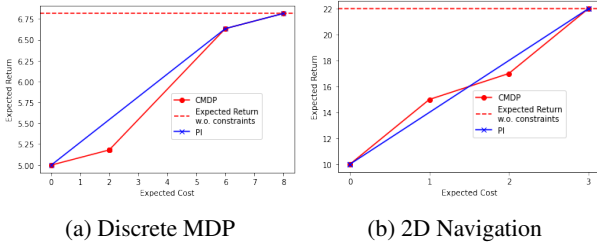

(a) Discrete MDP   (b) 2D Navigation

Figure 6: Policies found during optimization: x-axis denotes the expected cost of the policy while y-axis denotes the expected return. Red and blue lines represent our method (CMDP) and naive method (PI), respectively. In general, the expected return increases as the expected cost increases. We see that PI can only find a subset of intermediate policies while our method can find all.

We compare our sparse-explanation-constrained optimization procedure in Section 5 to a naive method, Policy Iteration (PI), in which we iteratively evaluate and improve the current policy until convergence. During policy improvement, we greedily choose the action with the largest gain.

In Figure 6, we see that our optimization procedure is able to find all intermediate policies that improve upon the current policy (in terms of reward) as well as the optimal solution (Figure 4, 5). Comparing to us, PI can only find some of the intermediate solutions. For 2D domain, PI can only find the optimal solution. Although the optimal policy collects the largest reward, the corresponding contrastive explanation is long (Section 6.2) even for a simple domain like our 2D

navigation. Our method allows users to select the policy based on their own preference for the interpretability of the contrastive explanation as well as their considerations of additional outcomes. For example, for the 2D Navigation, the users may prefer the policy when $\kappa = 2$: although it collects less rewards, it stays longer in the desired region, $0.2 \leq y < 0.3$ (Figure 5c).

7. Discussion and Conclusion

In this work, we build a framework that improves a current policy (with which the user is already familiar) such that the contrastive explanation given the current and improved policy will be sparse and human interpretable. Our contribution is two-fold: (1) we generate complete, user-friendly global contrastive explanations for two RL policies and (2) we optimize the current policy while explicitly constraining the length of the resulting contrastive explanation.

Compared to existing methods for generating contrastive explanations, our explanations are more complete while still being compact. For example, in van der Waa et al., the authors study the difference in the paths generated by rolling out two policies and only explain short-term outcomes. On our discrete MDP domain, their method would generate the following explanation:

"Starting from the initial state s_0, π_e will reach state s_1 and perform action 1, which visits the desired state s_2 but increases the trajectory length by 1. Then, π_e will reach state s_5 and perform action 1, which visits the desired state s_6 but increases the trajectory length by 1."

In Sukkerd et al., the authors explain the outcome difference without telling users why the difference occurs:

> "Starting from the initial state s_0, we can visit the desired states (s_2, s_6) more times by carrying out the alternate policy π_e. However, this would make the trajectory longer."

In contrast, our contrastive explanation is more compact and complete (see Section 4.4) as we explain the global behavioral differences of two policies and the resulting differences in their long term outcomes.

Our sparse-explanation-constrained policy optimization procedure allows the end-users to explicitly control the trade-off between improved returns, interpretability and other desirable outcomes. While sparse optimization has appeared across counterfactual explanation literature, we are, to our knowledge, first to consider sequential decision-making tasks under sparsity constraints applied to explanations.

In future works, we plan to apply our framework to large healthcare domains, with the potential of helping clinicians understand new treatment plans and potentially improving current plans.

8. Acknowledgement

References

Altman, E. *Constrained Markov decision processes: stochastic modeling*. Routledge, 1999.

Altmann, A., Toloşi, L., Sander, O., and Lengauer, T. Permutation importance: a corrected feature importance measure. *Bioinformatics*, 26(10):1340–1347, 2010.

Dash, S., Günlük, O., and Wei, D. Boolean decision rules via column generation. *arXiv preprint arXiv:1805.09901*, 2018.

Dolgov, D. A. and Durfee, E. H. Stationary deterministic policies for constrained mdps with multiple rewards, costs, and discount factors. In *IJCAI*, volume 19, pp. 1326–1331. Citeseer, 2005.

Doshi-Velez, F. and Kim, B. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

Došilović, F. K., Brčić, M., and Hlupić, N. Explainable artificial intelligence: A survey. In *2018 41st International convention on information and communication technology, electronics and microelectronics (MIPRO)*, pp. 0210–0215. IEEE, 2018.

Du, M., Liu, N., and Hu, X. Techniques for interpretable machine learning. *Communications of the ACM*, 63(1): 68–77, 2019.

Hanna, J. P., Stone, P., and Niekum, S. Bootstrapping with models: Confidence intervals for off-policy evaluation. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Hao, B., Ji, X., Duan, Y., Lu, H., Szepesvari, C., and Wang, M. Bootstrapping fitted q-evaluation for off-policy inference. In *International Conference on Machine Learning*, pp. 4074–4084. PMLR, 2021.

Jacovi, A., Swayamdipta, S., Ravfogel, S., Elazar, Y., Choi, Y., and Goldberg, Y. Contrastive explanations for model interpretability. *arXiv preprint arXiv:2103.01378*, 2021.

Karimi, A.-H., Barthe, G., Balle, B., and Valera, I. Model-agnostic counterfactual explanations for consequential decisions. In *International Conference on Artificial Intelligence and Statistics*, pp. 895–905. PMLR, 2020.

Karpathy, A., Johnson, J., and Fei-Fei, L. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

Kompa, B., Snoek, J., and Beam, A. L. Second opinion needed: communicating uncertainty in medical machine learning. *NPJ Digital Medicine*, 4(1):1–6, 2021.

Liu, G., Schulte, O., Zhu, W., and Li, Q. Toward interpretable deep reinforcement learning with linear model u-trees. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 414–429. Springer, 2018.

McGill, A. L. and Klein, J. G. Contrastive and counterfactual reasoning in causal judgment. *Journal of Personality and Social Psychology*, 64(6):897, 1993.

Miller, T. Explanation in artificial intelligence: Insights from the social sciences. *Artificial intelligence*, 267:1–38, 2019.

Puiutta, E. and Veith, E. Explainable reinforcement learning: A survey. In *International cross-domain conference for machine learning and knowledge extraction*, pp. 77–95. Springer, 2020.

Quinlan, J. R. Simplifying decision trees. *International journal of man-machine studies*, 27(3):221–234, 1987.

Rathi, S. Generating counterfactual and contrastive explanations using shap. *arXiv preprint arXiv:1906.09293*, 2019.

Rodriguez, I. D. J., Killian, T. W., Son, S.-H., and Gombolay, M. C. Interpretable reinforcement learning via differentiable decision trees. *CoRR*, abs/1903.09338, 2019. URL http://arxiv.org/abs/1903.09338.

Rudin, C., Chen, C., Chen, Z., Huang, H., Semenova, L., and Zhong, C. Interpretable machine learning: Fundamental principles and 10 grand challenges. *arXiv preprint arXiv:2103.11251*, 2021.

Sabour, S., Frosst, N., and Hinton, G. E. Dynamic routing between capsules. *Advances in neural information processing systems*, 30, 2017.

Spangher, A., Ustun, B., and Liu, Y. Actionable recourse in linear classification. In *Proceedings of the 5th workshop on fairness, accountability and transparency in machine learning*, 2018.

Sukkerd, R., Simmons, R., and Garlan, D. Tradeoff-focused contrastive explanation for mdp planning. In *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pp. 1041–1048. IEEE.

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

Thomas, P. and Brunskill, E. Data-efficient off-policy policy evaluation for reinforcement learning. In *International Conference on Machine Learning*, pp. 2139–2148, 2016.

van der Waa, J., Robeer, M., van Diggelen, J., Brinkhuis, M., and Neerincx, M. Contrastive explanations with local foil trees, 2018a.

van der Waa, J., van Diggelen, J., Bosch, K. v. d., and Neerincx, M. Contrastive explanations for reinforcement learning in terms of expected consequences. *arXiv preprint arXiv:1807.08706*, 2018b.

Verma, A., Murali, V., Singh, R., Kohli, P., and Chaudhuri, S. Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning*, pp. 5045–5054. PMLR, 2018.

Wachter, S., Mittelstadt, B., and Russell, C. Counterfactual explanations without opening the black box: Automated decisions and the gdpr. *Harv. JL & Tech.*, 31:841, 2017.

Yau, H., Russell, C., and Hadfield, S. What did you think would happen? explaining agent behaviour through intended outcomes. *CoRR*, abs/2011.05064, 2020. URL https://arxiv.org/abs/2011.05064.

Zhang, Q., Wu, Y. N., and Zhu, S.-C. Interpretable convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8827–8836, 2018.

A. Experiment Domain
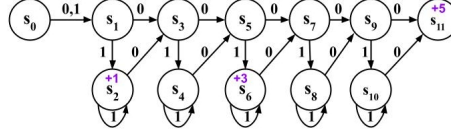
A.1. Toy Discrete MDP

Figure 7: Discrete MDP Definition

The discrete MDP has 11 states and binary actions ($|\mathcal{S}| = 11$, $\mathcal{A} \in \{0, 1\}$). s_{11} is the absorbing state with the initial state uniformly sampled from other states ($p_0(s) = \text{Cat}(\frac{1}{10}\vec{1})$). The transition dynamics $\mathcal{T}$ is defined as in Figure 7.

During explanation generation, users are interested in (a) the trajectory length (2) the number of visits to the desired states (s_2, s_6). During optimization, we aim to optimize the expected return with the reward function defined as

$$\mathcal{R}(s,a) = \begin{cases} 1 & \text{if } s = s_1, a = 1 \\ 3 & \text{if } s = s_5, a = 1 \\ 5 & \text{if } s \in \{s_9, s_{10}\}, a = 0 \\ -0.001 & \text{o.w.} \end{cases}$$
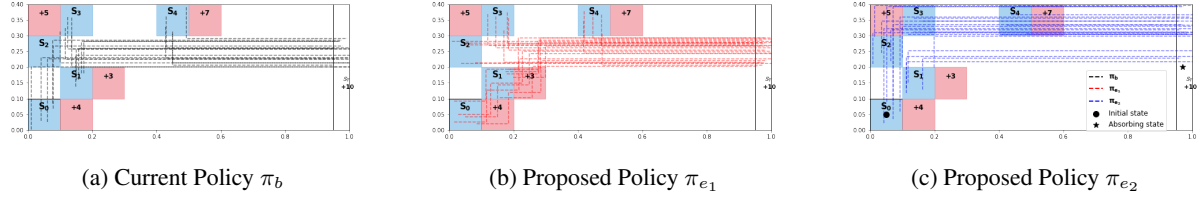
A.2. 2D Navigation

(a) Current Policy π_b (b) Proposed Policy π_{e_1} (c) Proposed Policy π_{e_2}

Figure 8: 2D Navigation Domain: Visualization of behaviors of current policy π_b and two proposed policies π_{e_1}, π_{e_2}. The blue boxes represent diverging regions with red boxes represent regions with positive rewards. π_b and π_{e_1} differ in region $\mathcal{S}_0$ and $\mathcal{S}_2$. π_b and π_{e_2} differ in region $\mathcal{S}_1$ and $\mathcal{S}_3$.

In the continuous 2D Navigation domain, the state space is defined by the coordinates $s = (x, y)$. The agent can either go east ($a = E$), north ($a = N$) or south ($a = S$). The initial state is uniformly sampled ($x \sim \text{Unif}(0, 0.1), y \sim \text{Unif}(0, 0.1)$) and the agent aims to navigate to the goal region ($x > 0.95$). The transition dynamics is defined as

$$\begin{cases} x' = x + 0.1 & \text{if } a = E \\ y' = y + 0.1 & \text{if } a = N \\ y' = y - 0.1 & \text{if } a = S \end{cases}$$

The reward function is defined as

$$\mathcal{R} = \begin{cases} 4 & \text{if } x \in [0.1, 0.2] \text{ and } y \in [0, 0.1] \\ 3 & \text{if } x \in [0.2, 0.3] \text{ and } y \in [0.1, 0.2] \\ 5 & \text{if } x \in [0, 0.1] \text{ and } y \in [0.3, 0.4] \\ 7 & \text{if } x \in [0.5, 0.6] \text{ and } y \in [0.3, 0.4] \\ 10 & \text{if } x > 0.95 \\ -0.001 & \text{o.w.} \end{cases}$$

The current policy π_b and two proposed policies π_{e_1}, π_{e_2} (Figure 8) is defined as

$$\pi_b(a|s) = \begin{cases} S & \text{if } x \in [0.1, 0.2] \cup [0.5, 0.6] \\ & \text{and } y > 0.3 \\ N & \text{else if } y < 0.2 \\ E & \text{o.w.} \end{cases}$$

$$\pi_{e_1}(a|s) = \begin{cases} E & \text{if } x \in [0, 0.1] \\ & \text{and } y \in [0., 0.1] \\ E & \text{else if } x \in [0.1, 0.2] \\ & \text{and } y \in [0.1, 0.2] \\ S & \text{else if } x \in [0.1, 0.2] \cup [0.5, 0.6] \\ & \text{and } y > 0.3 \\ N & \text{if } y < 0.2 \\ E & \text{o.w.} \end{cases}$$

$$\pi_{e_2}(a|s) = \begin{cases} N & \text{if } x \in [0, 0.1] \text{ and } y \in [0.2, 0.3] \\ N & \text{else if } y < 0.2 \\ E & \text{o.w.} \end{cases}$$

B. Algorithms

B.1. Collect Diverging States (Batch)

Suppose we are given a batch of data $\mathcal{D}_b$, we collect the diverging state using the following algorithm

---
Algorithm 3 Batch_Diverging_State_Collection $(\pi_1, \pi_2, \mathcal{D}_b)$

1: Initialize $\mathcal{D}, \mathcal{I} = \{\}, k = 1$
2: **for** $s \in \mathcal{D}_b$ **do**
3:    $a = \pi_1(a|s), \; a_2 = \pi_2(a|s)$
4:    **if** $f(s, \pi_1, \pi_2) = 1$ **then**
5:      **if** key $(a_1, a_2) \notin \mathcal{I}$ **then**
6:        add key: $\mathcal{I}[(a_1, a_2)] = k, k = k + 1$
7:      **end if**
8:      $(s, \mathcal{I}[(a_1, a_2)]) \to \mathcal{D}$
9:    **else**
10:      $(s, 0) \to \mathcal{D}$
11:    **end if**
12: **end for**
13: return $\mathcal{X}, \mathcal{Y}$

B.2. Adapted Boolean Decision Rules via Column Generation (BDCG)

BDCG is a binary interpretable classifier that learns Boolean rules in either disjunctive normal form (DNF, OR-of-ANDs) or conjunctive normal form (CNF, AND-of-ORs). We adapt BDCG so that it can be used for our multi-classification tasks.

Suppose we are given a data set consisting of N samples $(\mathbf{x}_n, y_n)$ with labels $y_n = 1, \cdots, K$. Let the set $\{1, \cdots, N\}$ be partitioned into $\bigcup_{k=1}^{K} \mathcal{Z}_k$ where $\mathcal{Z}_k$ contains the indices of samples with the label $y_n = k$. Without loss of generality, we sort $\mathcal{Z}_k$ by the set size and assume that $|\mathcal{Z}_1| \leq |\mathcal{Z}_2| \leq \cdots \leq |\mathcal{Z}_k|$. The pseudo-code of the adapted BDCG is as follows

---
Algorithm 4 BDCG_for_Multi-classification $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$

1: Initialize $\mathcal{H} = \{\}$
2: **for** $k = 1, \cdots, k - 1$ **do**
3:    Recreate labels $\mathcal{Y}'$ ($y_n = 0$ if $n \in \mathcal{Z}_k$ and $y_n = 1$ otherwise)
4:    $h_k =$ BDCG$(\mathcal{X}, \mathcal{Y}')$
5:    $h_k \to \mathcal{H}$
6:    $\mathcal{X} = \mathcal{X} \backslash \{x_n : n \in \mathcal{Z}_k\}, \mathcal{Y} = \mathcal{Y} \backslash \{y_n : n \in \mathcal{Z}_k\}$
7: **end for**
8: return $\mathcal{H}$

C. Construct a discrete MDP for continuous domains

Suppose we have collected K diverging regions and denote them as $\{\mathcal{S}_1, \mathcal{S}_2, \cdots, \mathcal{S}_K\}$. We construct a discrete MDP where the state space is defined as $\left(\bigcup_{k=1}^{K} \mathcal{S}_k\right) \bigcup \mathcal{S}_T$ ($\mathcal{S}_T$ is the absorbing state region in the original MDP). The action space remains the same as the original MDP. Recall that given an initial state s_0, a diverging path is defined as the set of diverging regions visited in a roll-out and is denoted as, $\mathcal{P}(s_0) = \{s_{k_1}, s_{k_2}, \cdots, s_T\}$.

Given a set of trajectories generated by following the current policy π_b and the proposed policies $\{\pi_{e_j}\}_{j=1}^{J}$, we can estimate the new transition dynamics as

$$\mathcal{T}(s_{k'}|s_k, a) = \frac{\sum \mathbb{1}_{\{s_{k_t} = s_k, a_t = a, s_{k_{t+1}} = s_{k'}\}}}{\sum \mathbb{1}_{\{s_{k_t} = s_k, a_t = a\}}}$$

and the new reward function as

$$\mathcal{R}(s_k, a) = \frac{\sum \mathcal{R}_{s_k:,a}}{\sum \mathbb{1}\{s_{k_t} = s_k, a_t = a\}}$$

where $\sum \mathcal{R}_{s_k:,a}$ denotes the total of the sequence of rewards collected starting from the diverging region s_k until reaching the next diverging region or the absorbing region.