Constrained Reinforcement Learning Using Successor Representations

Anonymous authors
Paper under double-blind review

Abstract

Real-world Reinforcement Learning depends on the ability to formulate safety constraints into a policy. Unfortunately, current methods are hard to adapt to changes in the cost function introduced by, e.g., domain shift or obstacles moving over time. The lack of adaptability means that policies are too unflexible to deal with complex real-world conditions. We propose the Safe Deep Successor Representation (SafeDSR), a novel method that allows quick retraining of policies towards new cost structures by decoupling the dynamics, reward structure, and costs by introducing a single learnable weight matrix. This matrix can be updated in a supervised manner instead of having to adapt the whole network if the cost structure of the environment changes. We demonstrate this ability in a freely configurable environment and show that our method is competitive with the state of the art while being considerably more flexible.[1]

1 Introduction

Reinforcement Learning (RL) is a powerful method for complex world control and planning problems. Unfortunately, RL's real-world utility remains limited due to the inability to encode functional and safety constraints into the policy. As safety cannot be naively enforced on a policy level in standard RL formulations, focus has shifted towards an extension of the normal Markov Decision Process (MDP) towards the Constrained Markov Decision Process (CMDP) (Altman, 1999).

To achieve this, the CMDP augments the MDP five-tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ (Sutton & Barto, 2018) with (potentially multiple) cost functions $C_j(s, a)$ and a cost budget b_j for $j \in \{1, \dots, J\}$. The optimization problem solving the CMDP can then be formulated as

$$\max_{\pi} \mathbb{E}_{p_\pi(\tau)} \left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t) \right] \quad \text{s.t.} \quad \mathbb{E}_{p_\pi(\tau)} \left[\sum_{t=0}^{T} \gamma^t C_j(s_t, a_t) \right] \leq b_j, \text{ for } j \in \{1, \dots, J\}, \tag{1}$$

where

$$p_\pi(\tau) = \mu(s_0) \prod_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t) \tag{2}$$

is the probability of a trajectory τ under policy π, and μ the initial state distribution.

One noticeable limitation of Equation (1) is that the costs C_j are considered fixed. This makes modeling dynamic or changing environments challenging since adapting to a new constraint (e.g., after a robot is moved to a different position) involves retraining the policy from scratch, even if the objective reward $R(s_t, a_t)$ is fixed. The reasons for allowing quick updates to constraints goes beyond the robotics scenario into, for instance, power-grid management (Chatzivasileiadis, 2018) (plants are installed or taken offline) or workforce scheduling (Birgin et al., 2014) (people get hired or skills change).

In this paper, our aim is to solve this problem using the concept of successor representations (Dayan, 1993) to explicitly model the reachability $M(s, a, s')$ of a state s' starting from any state s after action a was executed.

[1] The source code will be made publicly available upon acceptance.

This allows for the full decomposition of dynamics and reward/constraints. Changing the constraints (or rewards for that matter) only means re-weighting the reachability with a new cost or reward function, which is considerably easier to train. Our main contributions can be summarized as follows:

- We propose the Safe Deep Successor Representation (SafeDSR), which leverages the decoupling of dynamics and rewards of the Deep Successor Representation (DSR) (Kulkarni et al., 2016) to model the costs of CMDPs independently of their reward structure.
- We demonstrate that, while remaining competitive with state-of-the-art methods (Section 5.3), this decoupling enables SafeDSR to adapt to changes in the cost distribution within a few environment interactions (Section 5.4).
- For this purpose, we develop a freely configurable constrained grid environment with a continuous state space, allowing for changes in the cost distribution during training.
- We introduce the Safe Deep Q-Network (SafeDQN) as a baseline and for comparison to SafeDSR, as both are value-based methods that incorporate constraints using Lagrangian relaxation.
- We motivate the use of the Safe Goal Count and the Average Safe Goal Rate as metrics for comparing algorithms in Constrained Reinforcement Learning (CRL), each offering a single, interpretable evaluation metric that avoids the need to balance multiple objectives.

We structure the remainder of the article as follows. After discussing related work (Section 2), we provide background to Lagrangian relaxations of CMDPs, Q-learning and the successor representation (Section 3). Next, we will introduce our two methods: First, we showcase a SafeDQN model without using the successor representation. Second, we introduce our SafeDSR to compare against the naive Lagrangian relaxation provided by SafeDQN (Section 4). Finally, we compare our methods against state-of-the-art CMDP algorithms on a continuous long-context credit assignment grid world challenge (Section 5).

2 Related Work

CRL has historically relied on fixed cost and budget formulations. One of the most seminal works on modern Deep Constrained RL is Constrained Policy Optimization (CPO) (Achiam et al., 2017). This extends the popular TRPO algorithm (Schulman et al., 2015) towards handling external constraints. Stooke et al. (2020) consider combining existing methods like PPO or TRPO with a Lagrangian parameter tuned using PID controllers. We follow (Ji et al., 2024) and call these CPPO-PID and TRPO-PID. Interior-point Policy Optimization (IPO) (Liu et al., 2019) considers using interior-point methods to guarantee safe policies remain safe. Reward Constrained Policy Optimization (RCPO) (Tessler et al., 2018) uses a multi-timescale approach as an alternative objective guiding the policy towards the feasible set. Penalized Proximal Policy Optimization (P3O) (Zhang et al., 2022) eliminates constraints by penalizing and clipping the objective to obtain an unconstrained problem. However, while all these methods allow to introduce constraints, they learn to estimate the future expected cost and therefore require full trajectory rollouts to adapt to changes in the cost structure. In contrast, our method separates the dynamics from the cost estimation, allowing updates only to the single-step cost estimator.

On the side of flexibility, the Budgeted Markov Decision Process (BMDP) (Carrara et al., 2019) allows one to change the constraint budget b dynamically without retraining, while keeping the constraint function itself fixed. In contrast, our method is designed to quickly adapt to changes in the cost function $C(s, a)$. This does require some additional training steps (since the new cost $C(s, a)$ can have any arbitrary unknown form), but allows for significantly more flexibility.

We base our SafeDQN method on the original Deep Q-Network (DQN) proposed by Mnih et al. (2015). Our SafeDSR algorithm is based on Deep Successor RL (Kulkarni et al., 2016) and the subsequent work by Barreto et al. (2017). The focus of these two works was dynamic adaptation towards different environments, without considering constraints. This work motivates the usage of such successor representations in CRL.

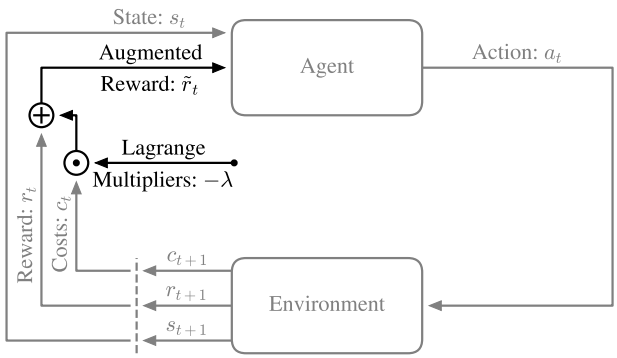

Figure 1: **Agent-environment interface of the CMDP when using Lagrangian relaxation.** By combining the reward r_t and costs c_t using Lagrange multipliers λ into the cost-augmented reward $\tilde{r}_t$ (black), the CMDP (gray) appears as a MDP from the agent's perspective. Note that λ may change over time.

3 Background

3.1 Lagrangian Relaxation

As demonstrated in libraries such as OmniSafe (Ji et al., 2024), a common method for finding constraint-satisfying policies $\pi \in \Gamma$ is to formulate the CMDP as a constrained optimization problem and solving it using Lagrange multipliers $\lambda \in \mathbb{R}_+^J$:

$$\max_{\pi} \min_{\lambda \geq 0} \mathbb{E}_{p_\pi(\tau)} \left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t) - \lambda \cdot \left(\sum_{t=0}^{T} \gamma^t C(s_t, a_t) - b \right) \right] \tag{3}$$

This formulation allows solving CMDPs using methods designed for solving unconstrained MDPs by training those on cost-augmented rewards $\widetilde{R}(s_t, a_t) = R(s_t, a_t) - \lambda \cdot C(s_t, a_t)$, as depicted in Figure 1, while in parallel updating the Lagrangian parameters using gradient descent. This approach is guaranteed to converge to the optimal policy in the limit of infinite data due to CMDPs having a zero-duality gap (Paternain et al., 2019).

3.2 Q-learning

Sequential decision problems can be solved by iteratively updating the state-action value function

$$Q^\pi(s, a) = \mathbb{E}_{p_\pi(\tau)} \left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t) \middle| s_0 = s, a_0 = a \right] \tag{4}$$

using temporal-differences and by improving the policy through acting greedily on the current Q-value estimates (Watkins & Dayan, 1992). In many practical scenarios, the Q-function needs to be approximated using deep neural networks because of too extensive state- and action-spaces. This algorithm is called DQN (Mnih et al., 2015) and uses a target-network and experience replay to stabilize the training.

3.3 Successor Representation

Alternatively, we can learn the successor representation

$$M^\pi(s, a, s') = \mathbb{E}_{p_\pi(\tau)} \left[\sum_{t=0}^{T} \gamma^t \mathbb{I}(s_t = s') \middle| s_0 = s, a_0 = a \right], \tag{5}$$

which measures the expected (discounted) state occupancy (Dayan, 1993). Here, $\mathbb{I}(x = y)$ is the indicator function, which is 1 if $x = y$ and 0 otherwise. $M^\pi(s, a, s')$ measures how often the state s' is reached when

choosing action a being in state s. It allows rewriting the Q-function using a reward predictor $r(s)$:

$$Q^\pi(s,a) = \sum_{s' \in \mathcal{S}} M^\pi(s,a,s') r(s'). \tag{6}$$

In practice, the successor representation needs to be approximated using deep neural networks as well. For this purpose, Kulkarni et al. (2016) proposed to learn a reward predictor

$$R(s) \approx \phi(s) \cdot w_r \tag{7}$$

using features $\phi(s) \in \mathbb{R}^d$ and reward weights $w_r \in \mathbb{R}^d$, where d is the feature dimension. This assumption is without loss of generality, since any component of $\phi(s)$ can learn to replicate the reward $r(s)$. Those features $\phi(s)$ allow rewriting the Q-function as

$$Q^\pi(s,a) \approx \underbrace{\mathbb{E}_{p_\pi(\tau)} \left[\sum_{t=0}^{T} \gamma^t \phi(s_t) \middle| s_0 = s, a_0 = a \right]}_{:= \text{ successor features } M^\pi(s,a)} \cdot w_r. \tag{8}$$

DSR (Kulkarni et al., 2016) learns features ϕ and reward weights w_r using a reward and reconstruction loss:

$$L_{\text{features}} = \underbrace{\big(R(s) - \phi(s) \cdot w_r\big)^2}_{\text{reward loss}} + \underbrace{\big|\big|s - g(\phi(s))\big|\big|_2^2}_{\text{reconstruction loss}}. \tag{9}$$

Here, g is an inverse feature network. The successor features M are updated using an adapted DQN loss

$$L = \Big(R(s) + \gamma Q_{\text{prev}}(s',a') - Q(s,a)\Big)^2 \approx \Big|\Big|\phi(s) + \gamma M_{\text{prev}}(s',a') - M(s,a)\Big|\Big|_2^2, \tag{10}$$

where the transition (s,a,s') is sampled from a replay buffer, the action a' is the best action selected using the main network, and $M_{\text{prev}}(s',a')$ is the target network. The parameters of the reward, feature, and inverse feature network are updated in a supervised manner and independently of the parameters of the successor features M. Additionally, a biased sampling approach is used to oversample seldomly occurring rewards.

4 Methods

We propose SafeDSR as our main algorithm for solving CMDPs. In addition, we define SafeDQN as a baseline for comparison with SafeDSR, since DSR (Kulkarni et al., 2016) builds upon DQN (Mnih et al., 2015). Both algorithms combine a value-based method with Lagrangian relaxation to incorporate cost constraints.

4.1 Simple Baseline: Safe Deep Q-Network (SafeDQN)

The SafeDQN algorithm trains the Q-network on cost-augmented rewards $\widetilde{R}(s_t,a_t) = R(s_t,a_t) - \lambda \cdot C(s_t,a_t)$, where the costs are scaled with Lagrangian parameters λ, as sketched in Figure 1. Those parameters are tuned with a proportional controller which increases them if the corresponding constraint is violated at the end of an episode and decreases them otherwise. A detailed description of this algorithm can be found in Appendix A and the used parameters, which are based on the DQN implementation by Huang et al. (2022), in Appendix B. The issue with this approach is that the Q-network learns both the dynamics and the cost-augmented reward, where the latter depends on the Lagrangian parameters. Therefore, a large change in those parameters requires the whole Q-function to be adapted, which is a slow process.

There are multiple possible solutions to this problem. For instance, Schmidt et al. (2022) proposed to learn a Q-function for the future expected reward and a future expected cost predictor K and only use the Lagrangian parameters during decision time. However, in this case all predictors learn the dynamics of the environment, which is computationally expensive and unnecessary because the Q- and K-functions share the same dynamics. Further, even if the dynamics stay constant, changing the cost or reward structure requires complete retraining of the Q- and K-functions.

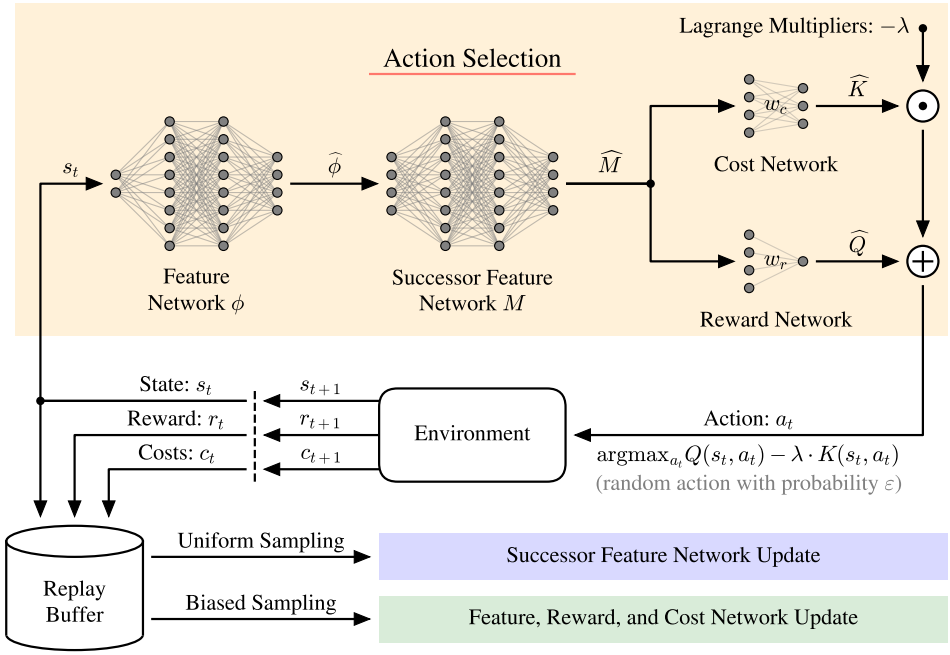

Figure 2: **Overview of the action selection process of the SafeDSR agent.** The agent interacts with the environment, receiving states s_t, rewards r_t, and costs c_t, which are stored in a replay buffer. The different networks are trained by sampling mini batches from the replay buffer. The successor features M are hereby optimized independently of the feature network ϕ and the reward and cost networks. The training processes for both are outlined in Appendix C. To select an action a_t, the agent calculates the future expected state occupancy $\widehat{M}$ embedded in a high-dimensional space for each possible action a_t in the current state s_t. The future expected reward $\widehat{Q} := Q(s_t, a_t)$ and cost $\widehat{K} := K(s_t, a_t)$ are then computed by applying the reward and cost networks to the successor features $\widehat{M}$. This effectively accounts for the reward and costs that occur along the estimated future trajectory. The next action a_t is then selected by choosing the action that maximizes the future expected cost-augmented reward $Q(s_t, a_t) - \lambda \cdot K(s_t, a_t)$, or by choosing a random action with a probability ε. The Lagrange multipliers λ are optimized with a proportional controller.

4.2 Safe Deep Successor Representation (SafeDSR)

Hence, we use a different approach, the SafeDSR. The DSR allows decoupling the reward and the dynamics by using successor features M. This formulation of Q-learning is predestined to work with Lagrangian relaxation because the cost-augmented reward is a linear combination of rewards and costs. For this purpose, we generally follow Kulkarni et al. (2016) in our DSR implementation with a few modifications to learn safe policies.

In contrast to Kulkarni et al. (2016), we consider the reward to be state-action-dependent rather than just state-dependent. Hence, the features ϕ depend on the state-action pairs (s, a). We model this by concatenating the states s with the one-hot encoded actions a. In addition to the successor feature network M, we also use a target network ϕ_{prev} for our features ϕ to address the moving target in Equation (10). This leads to the successor feature loss

$$L = \left|\left|\phi_{\text{prev}}(s, a) + \gamma M_{\text{prev}}(\phi_{\text{prev}}(s', a')) - M(\phi(s, a))\right|\right|_2^2, \tag{11}$$

which is outlined in Appendix C.[2] Further, we normalize the feature vectors ϕ to $||\phi(s, a)||_2^2 = 1$. This ensures more stability in practice, due to a consistent input range for the subsequent networks. Additionally,

[2]Kulkarni et al. (2016) already used the features ϕ as input for the successor features M.

Kulkarni et al. (2016) implement the bias sampling approach for training the features ϕ and the reward predictor using a secondary buffer containing non-zero reward samples. They draw 20% of the samples from this secondary buffer and 80% from the replay buffer. In contrast, we obtain the 20% by drawing $\frac{20\%}{J+1}$ of samples from all past experiences with probabilities inversely proportional to the reward and to each of the J cost distributions. Finally, we freeze the weights of the feature network ϕ after t_{freeze} environment steps, as proposed by Barreto et al. (2017), which stabilizes the training by fixing the input distribution for the successor network M.

To extend the DSR algorithm to work with costs, we add a cost predictor

$$C(s,a) \approx w_c \cdot \phi(s,a), \tag{12}$$

where $w_c \in \mathbb{R}^{J \times d}$ contains the parameters of the cost predictor network. This allows to approximate the future expected costs as

$$K^\pi(s,a) \approx w_c \cdot \underbrace{\mathbb{E}_{p_\pi(\tau)}\left[\sum_{t=0}^{T} \gamma^t \phi(s_t,a_t) \middle| s_0 = s, a_0 = a\right]}_{:= \text{ successor features } M^\pi(s,a)}. \tag{13}$$

Hence, we need to extend the feature loss Equation (9) to

$$L_{\text{features}} = \underbrace{\big(R(s,a) - \phi(s,a) \cdot w_r\big)^2}_{\text{reward loss}} + \underbrace{\big|\big|C(s,a) - w_c \cdot \phi(s,a)\big|\big|_2^2}_{\text{cost loss}} + \underbrace{\big|\big|(s,a) - g(\phi(s,a))\big|\big|_2^2}_{\text{reconstruction loss}}. \tag{14}$$

Here, (s,a) are the states s concatenated with the one-hot encoded actions a. This equation is also depicted in Appendix C.

Finally, to learn safe policies, we adapt the agent's action selection rule using Lagrangian relaxation:

$$\arg\max_a \left\{ Q^\pi(s,a) - \lambda \cdot (K^\pi(s,a) - b) \right\}. \tag{15}$$

An overview of the action selection process is depicted in Figure 2. It differs from the naive Lagrangian relaxation approach in Figure 1, which is used in SafeDQN, by using the Lagrangian parameters only for decision-making, and not for learning future estimates.

We tune the Lagrangian parameters using a proportional controller according to Equation (16) at the end of each episode, where α_λ is the learning rate and C_{ep} the episode cost. This approach is equivalent to optimizing Equation (3) via gradient descent:

$$\lambda \leftarrow \max(0, \lambda + \alpha_\lambda (C_{\text{ep}} - b)). \tag{16}$$

A detailed description of the SafeDSR algorithm can be found in Appendix A and the used parameters in Appendix B.

5 Experiments

In the following, we describe the conducted experiments. In all of them, we use the parameters listed in Appendix B for SafeDQN and SafeDSR. For the other algorithms, we use the implementation and parameters from OmniSafe (Ji et al., 2024), specifically the experimental version developed by Zhou (2023) for discrete action spaces. Further, we always use five seeds, a discount factor of $\gamma = 0.99$, and a cost budget of 5.

5.1 Continuous Grid Environment

We consider a goal seeking task in a grid environment, as shown in Figure 3. The dynamics of this environment are based on Barreto et al. (2017) and were extended to a CMDP by adding costs.

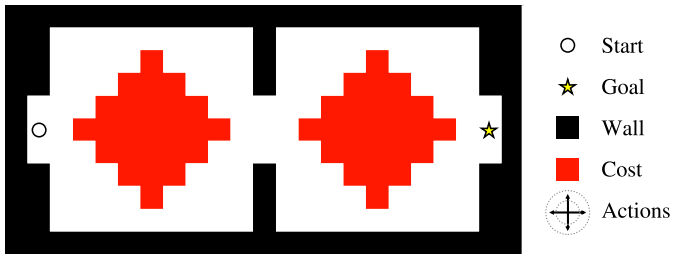

Figure 3: **Constrained grid environment with a continuous state space.** The agent can walk in the cardinal directions with a noisy step length. An episode ends if the agent reaches the goal. Additionally, the agent incurs a cost of 1 for each step taken inside a cost cell.

The states s_t are the two-dimensional positions of the agent in the maze. They are defined as $s_t = (\alpha x_t, \alpha y_t)$, where $x_t \in [0, x_{\max}] \subset \mathbb{R}$, $y_t \in [0, y_{\max}] \subset \mathbb{R}$ and $\alpha^{-1} = \max(x_{\max}, y_{\max})$. The rescaling with α is done to transfer the state values into a more sensible range for function approximation. The agent can move in the cardinal directions using the discrete actions up, down, left, and right. These actions have a noisy step length drawn from the normal distribution $\mathcal{N}(\mu = 0.75,\ \sigma = 0.075)$, clipped to $\mu \pm 3\sigma$ to simplify collision handling with the walls. If an action sets the agent inside a wall, the action is undone. We use a deterministic reward function where the agent receives a reward of 1 when reaching the goal cell and a time penalty of -0.01 for every step taken. This CMDP contains only a single constraint ($J = 1$), and thus only a single Lagrangian parameter is needed. The agent receives a cost of 1 for each step taken inside a cost cell. The described task ends when the agent reaches the goal. Additionally, we truncate if the agent takes more than 1000 environment steps, since our environment is solvable in fewer than 100 steps, allowing the agent to return to the start if it has gone astray.

It is worth noting that this environment does not fully represent the challenges addressed by modern RL techniques. This environment is similar to common benchmark environments in CRL, for example, the goal task in Safety-Gymnasium (Ji et al., 2023), but with a lower-dimensional state space and discrete actions. The low-dimensionality simplifies plotting of the trajectories and the learned value functions, which facilitates debugging and interpretation. Further, this environment allows for varying the exploration challenge by increasing or decreasing the number of rooms, as shown exemplarily in Appendix D.

5.2 Metrics

Typical metrics in CRL include the **future expected reward and cost**, or a variation thereof. The best performing algorithms have a high future expected reward and a low future expected cost at the end of training. However, it is generally unclear and task-dependent whether a slightly larger future expected cost is acceptable for an increase in the future expected reward.

Hence, we consider the **goal and safe goal count**, where the former measures the number of all trajectories during training which reach the goal and the latter only measures trajectories which reach the goal without violating any constraints. Both metrics are monotonically increasing over the training time and more stable than the average episodic reward. We call the derivatives of the goal and safe goal count, the **goal and safe goal rate**. The averages of those rates describe the likelihood of the agent reaching the goal.

We focus on these metrics to compare the different algorithms in our experiments. If the metrics are shown as a time series, they are first linearly interpolated to obtain support points every 200 environment steps and then averaged over the different runs. To obtain a trend line, for instance, the average goal and safe goal rate, we apply a right-aligned window average with a window size of 20 000 environment steps afterwards. If we only discuss a metric at the end of training, we average over the last recorded event for each seed.

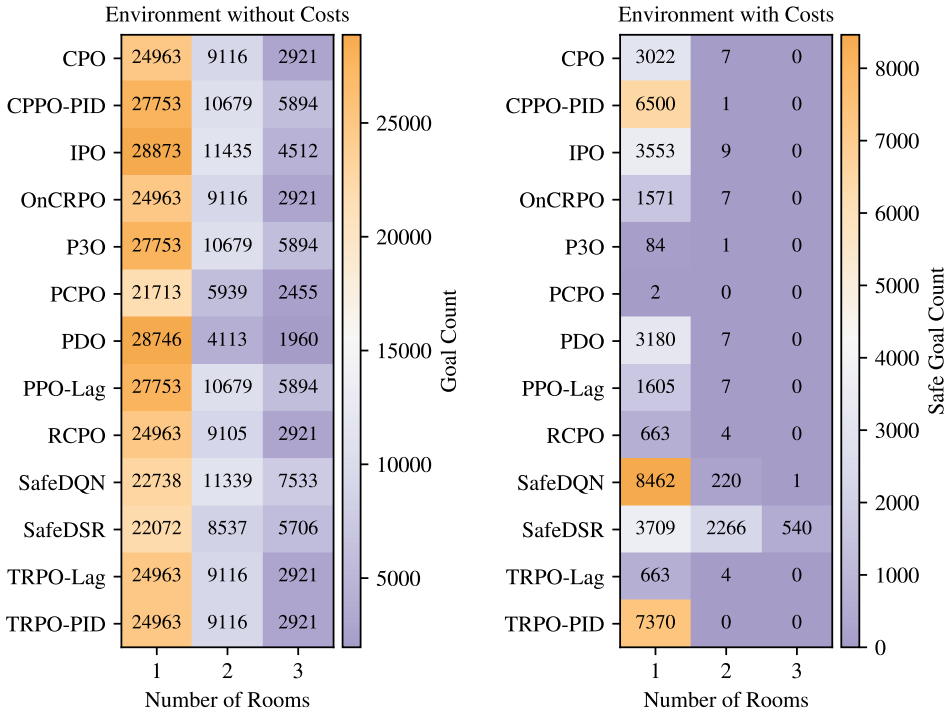

Figure 4: **Goal and safe goal count for the different baseline algorithms, SafeDQN, and SafeDSR** after 500 000 environment steps in environments with and without costs. The counts were averaged over five seeds and rounded to the nearest integer. On the left, each algorithm was run in the environments, as show in Appendix D without costs. On the right, the same algorithms were used with a cost budget of 5 in the same environments with costs. All algorithms are able to reach the goal in the costless environments. Most are even able to reach a goal count larger than SafeDSR in the environments with one or two rooms. In contrast, only CPPO-PID, SafeDQN, and TRPO-PID outperform SafeDSR in the cost environments when using a single room. In the two- and three-room cost environments SafeDSR outperforms all other algorithms.

5.3 Baseline Comparison

In this section, we compare SafeDSR with SafeDQN and several baseline algorithms implemented in OmniSafe (Ji et al., 2024). Figure 4 summarizes the main results of this experiment. In the one- and two-room environment without costs, many algorithms are capable of reaching the goal more often than SafeDSR. However, when costs are introduced in environments with multiple rooms, only SafeDSR consistently reaches the goal safely. Because the safe goal count does not show how fast and consistently the algorithms find safe trajectories, we show the average safe goal rate in Figure 5.

In the one-room environment, many algorithms eventually learn safe policies. While the seven best-performing algorithms reliably locate the goal after approximately 250 000 environment steps, their average safe goal rates vary significantly. For example, the PID-Lagrangian methods achieve average safe goal rates of over 70% by the end of training, whereas SafeDSR and SafeDQN settle at around 60%. This implies that the naive proportional controller from Equation (16) might be improved by incorporating integral and derivative terms. Other algorithms, such as CPO, show promising performance mid-training but eventually fail to maintain safe trajectories.

The situation differs in the two- and three-room environments. Where only SafeDSR reaches an average safe goal rate over 60% and 30%, respectively. This observation comes with the caveat that SafeDQN can reach a much higher average safe goal rate in the two-room environment given enough training steps. The longer training time for SafeDQN is likely due to the implicit handling of the Lagrangian parameters

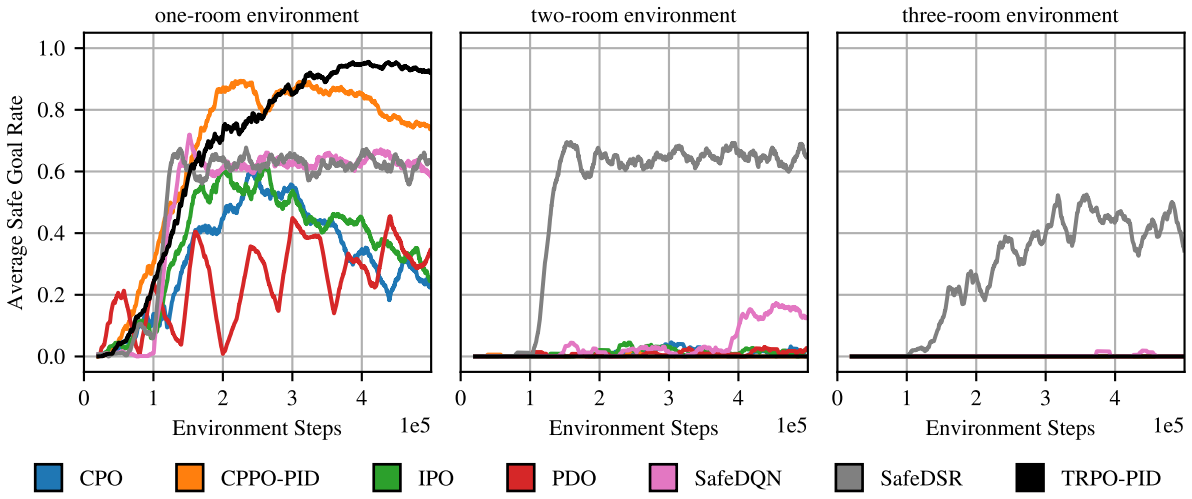

Figure 5: **Average safe goal rate for different baseline algorithms, SafeDQN, and SafeDSR.** We only show the seven best-performing algorithms from Figure 4. The PID-controlled algorithms, CPPO-PID and TRPO-PID have the largest safe goal rates at the end of training in the one-room environment, while SafeDQN and SafeDSR perform slightly worse. In the other environments, only SafeDSR is able to consistently find safe trajectories.

and costs compared to SafeDSR which explicitly learns the cost distribution. In contrast, we found in additional experiments that even five-times as many environment steps did not improve the performance of the other algorithms significantly. Further, delaying the start of the Lagrangian parameter tuning for the four Lagrangian methods PDO, RCPO, TRPO-Lag and PPO-Lag did only marginally change their performance, compared to SafeDSR where a delay facilitates exploration and improves its performance.

We suspect that many of the tested algorithms are limited by their exploration strategies, which is hinted by the trajectories shown in Figure 6 for the two-room environment. Many algorithms get either stuck in the corners of the environment, because of an overcorrecting behavior towards the cost cells (e.g., CPO), or get stuck in local optima (e.g., RCPO). In contrast, both the SafeDSR and SafeDQN agent walk around the cost cells, reaching the goal. This is very likely due to their high exploration rate ($\varepsilon = 0.25$), which increases the likelihood of finding cost-avoiding paths through the second room even if the Lagrangian parameter tuning overcorrected to avoid the cost cells.

We acknowledge that these results are not without limitations, since we use the default parameters for the used OmniSafe algorithms, while the SafeDSR algorithm was tuned for this work and SafeDQN was adapted appropriately. Hence, the OmniSafe implementations might underperform compared to SafeDQN and SafeDSR. However, by using eleven different algorithms from OmniSafe, we already test a wide range of parameter sets, where many outperform SafeDSR in the one- and two-room environment without costs but underperform in the environments with costs.

5.4 Changing Cost Distribution

Finally, we discuss the model's ability to respond to changing cost distributions. For this, we consider a policy trained on the "diamond" shape shown in Figure 3 and adapt it towards a different shape. We consider three different methods of doing this adaptation: First, one can only change the cost weights w_c without changing the Lagrangian ("post-adaptation" evaluation). This, arguably, is the minimal amount of change necessary to adapt towards an unforeseeable change in cost. In addition, we may also tune the Lagrangian parameter λ, since a change in cost distribution generally also changes the intensity necessary to repel dangerous states. For this, we adapt just the Lagrangian in an on-policy fashion ("penalty" evaluation). Finally, we consider training the full model — including reward, cost, Lagrange parameters, and successor features — for an

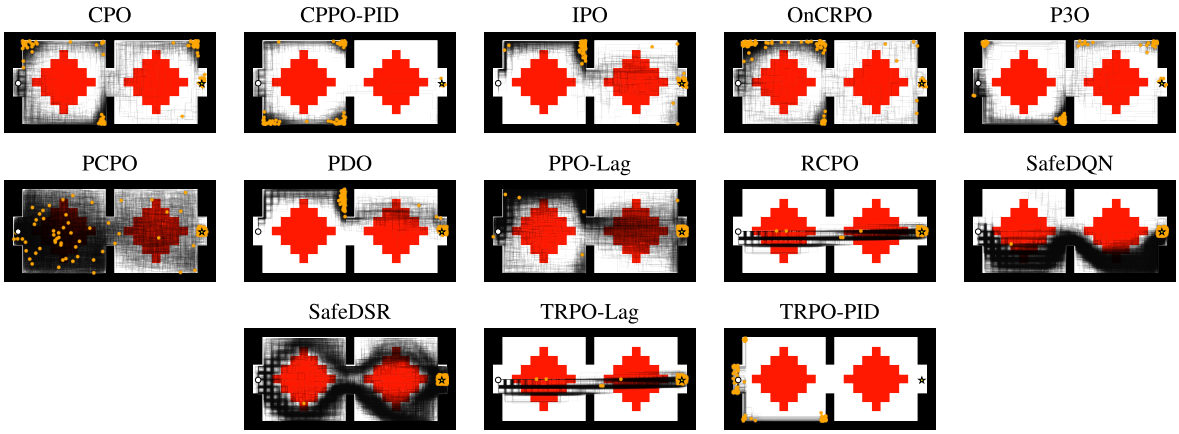

Figure 6: **Trajectories the different baseline algorithms, SafeDQN, and SafeDSR** have observed during the last 100 000 environment steps in the two-room environment. In these plots, we show the trial with the largest safe goal count for each algorithm. We use the goal count as a tie-breaker. Many algorithms are not able to reach the goal safely because they are unable to circumvent the wall separating the rooms. Meanwhile, RCPO and TRPO-Lag got stuck in local optima.

additional 100 000 gradient steps to demonstrate the theoretically achievable improvement if we didn't take advantage of the factorization into dynamics, reward, and cost models. Note that this is generally necessary if the policy changes too much.

One important consideration when using successor features to adapt towards changing cost and reward functions is that previously uninteresting states may become interesting after the change. Especially in the case of CRL this might lead to states that were entirely unexplored due to being hidden behind the constraints. For this reason, we consider a "pretraining" stage where *only* the feature encoder is pretrained using Equation (14) for random rollouts on the continuous grid.[3] In a real-world navigation setting, this would correspond to, e.g., adding random images from an empty warehouse to the dataset to allow for good feature embeddings. We argue that this is a small assumption in practice, as good pretrained models are generally available and can be used as feature embedders (Rahbar et al., 2022).

We showcase the empirical paths walked by our methods similar to Figure 6 since a direct comparison of rewards/costs is rather fraught: Changing the environment cost function can make the underlying problem harder or easier to an arbitrary level, so comparing achieved rewards is not meaningful. We supply the average safe goal rate during the different training phases in Appendix F.

We compare two adaptation settings "Diamond → Square" and "Diamond → Inverse Diamond" where the initial "diamond" shape gets adapted into either a large "square" or gets "inverted" such that all feasible areas become infeasible and vice versa (with small connections towards the start and end, see Figure 8). The first setting is used to test the impact of choosing a harder cost distribution than the initial one (since there is significantly less feasible area), while the second setting is intended to test an "adversarial example" where the configuration is built to force the agent to walk through exactly those areas that were infeasible before.[4]

We show our results in Figure 7, as one can see our method manages to quickly adapt its feasible region towards the new "square" cost structure, even if only the reward and cost function is trained. We also generally see the method with pretraining outperforming the one without, especially once the successor representation itself is trained. This aligns well with prior research on successor representations that highlight the crucial importance of the underlying features.

[3]Note that this means we do not have any dynamics information in our dataset. The pretraining information is just a set of random observations!

[4]We provide an additional experiment for a scaled "diamond" in Appendix E.

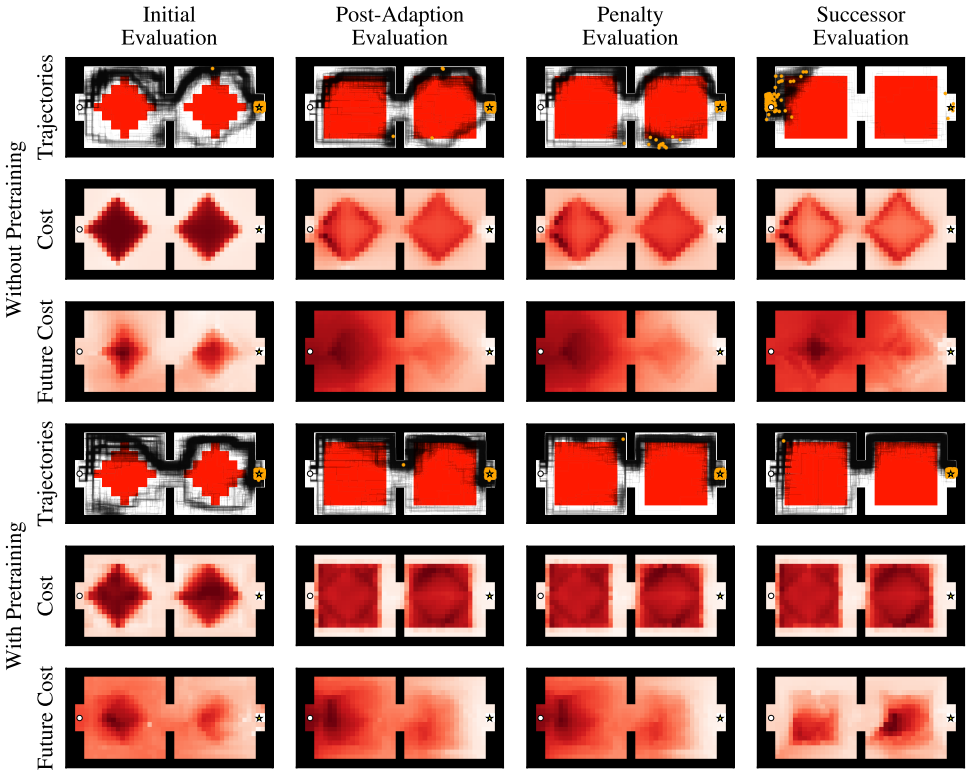

Figure 7: **Translation from a model pretrained on the "diamond" shape (Figure 3) towards an unseen "square" distribution.** We refer to training reward and cost functions as "post-adaptation" evaluation, to training the Lagrangian on-policy as "penalty" evaluation, and finally to training the full network as "successor" evaluation. We also show the performance of our method with or without pretraining the feature embedding model.

Looking at the adversarial example case in Figure 8, we find that even in such a worst-case scenario, our method manages to find a feasible policy despite the original policy having the exact opposite movement pattern. We observe that while adapting the cost function itself generally improves the results in all settings, the adaptation of the penalty coefficient and full model training has varying impact: Pretrained models tend to benefit from full model training, while for non-pretrained models performance may actually decrease. We hypothesize that this is due to the fact that there are no good features for the previously constrained areas, which makes the problem partially observable from the point of view of the feature embeddings. The pretrained model has no such dearth of data in the constrained areas, meaning it can learn a good successor model. Nevertheless, both the pretrained and non-pretrained model manage to find solutions to the path finding problem.

6 Conclusion

We propose a new method for Constrained Reinforcement Learning that allows quick adaptation towards novel cost functions by decomposing the dynamics and the associated costs and rewards. Therefore, our method can respond to unforeseen changes in reward or cost by just changing the reward and cost components, but keeping the dynamics fixed.

We demonstrate our model's ability to act in complex environments by comparing it to existing CRL methods on an interpretable stochastic continuous grid world with freely configurable costs. Our method outperforms existing CRL solvers considerably. Finally, we show that our method can learn to adapt to new constraints quickly by adapting a model pretrained on a "diamond" constraint form to different constraint formulations.

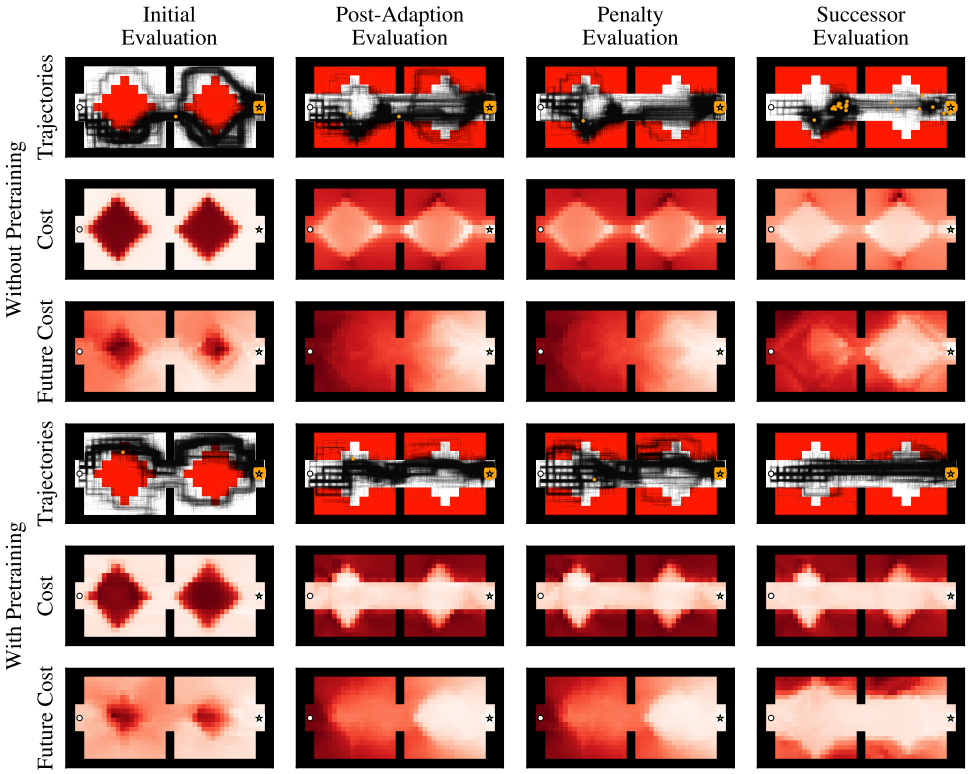

Figure 8: **Translation from a model pretrained on the "diamond" shape (Figure 3) towards an unseen "inverse diamond" distribution.** We refer to training reward and cost functions as "post-adaptation" evaluation, to training the Lagrangian on-policy as "penalty" evaluation, and finally to training the full network as "successor" evaluation. We also show the performance of our method with or without pretraining the feature embedding model.

References

Joshua Achiam, David Held, Aviv Tamar, and P. Abbeel. Constrained policy optimization. *ArXiv*, abs/1705.10528, 2017.

Eitan Altman. *Constrained Markov Decision Processes: Stochastic Modeling*. Routledge, December 1999. ISBN 9781315140223.

Andre Barreto, Will Dabney, Remi Munos, Jonathan J Hunt, Tom Schaul, Hado P van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Ernesto G. Birgin, Paulo Feofiloff, Cristina G. Fernandes, Everton L. de Melo, Marcio T. I. Oshiro, and Débora Pretti Ronconi. A milp model for an extended version of the flexible job shop problem. *Optimization Letters*, 8:1417–1431, 2014.

Nicolas Carrara, Edouard Leurent, Romain Laroche, Tanguy Urvoy, Odalric-Ambrym Maillard, and Olivier Pietquin. Budgeted reinforcement learning in continuous state space. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Spyros Chatzivasileiadis. Lecture notes on optimal power flow (opf). *ArXiv*, abs/1811.00943, 2018.

Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624, 1993.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

Jiaming Ji, Borong Zhang, Jiayi Zhou, Xuehai Pan, Weidong Huang, Ruiyang Sun, Yiran Geng, Yifan Zhong, Josef Dai, and Yaodong Yang. Safety gymnasium: A unified safe reinforcement learning benchmark. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 18964–18993. Curran Associates, Inc., 2023.

Jiaming Ji, Jiayi Zhou, Borong Zhang, Juntao Dai, Xuehai Pan, Ruiyang Sun, Weidong Huang, Yiran Geng, Mickel Liu, and Yaodong Yang. Omnisafe: An infrastructure for accelerating safe reinforcement learning research. *Journal of Machine Learning Research*, 25(285):1–6, 2024.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ArXiv*, abs/1412.6980, 2014.

Tejas D. Kulkarni, Ardavan Saeedi, Simanta Gautam, and Samuel J. Gershman. Deep successor reinforcement learning. *ArXiv*, abs/1606.02396, 2016.

Yongshuai Liu, J. Ding, and Xin Liu. Ipo: Interior-point policy optimization under constraints. In *AAAI Conference on Artificial Intelligence*, 2019.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Santiago Paternain, Luiz Chamon, Miguel Calvo-Fullana, and Alejandro Ribeiro. Constrained reinforcement learning has zero duality gap. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Arman Rahbar, Emilio Jorge, Devdatt Dubhashi, and Morteza Haghir Chehreghani. Do kernel and neural embeddings help in training and generalization? *Neural Processing Letters*, 55(2):1681–1695, 2022.

Lukas M. Schmidt, Sebastian Rietsch, Axel Plinge, Bjoern M. Eskofier, and Christopher Mutschler. How to learn from risk: Explicit risk-utility reinforcement learning for efficient and safe driving strategies. In *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1913–1920, 2022.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1889–1897, Lille, France, 2015.

Adam Stooke, Joshua Achiam, and P. Abbeel. Responsive safety in reinforcement learning by pid lagrangian methods. In *International Conference on Machine Learning*, 2020.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Bradford Books, Cambridge, MA, second edition, 2018. ISBN 9780262039246.

Chen Tessler, Daniel Jaymin Mankowitz, and Shie Mannor. Reward constrained policy optimization. *ArXiv*, abs/1805.11074, 2018.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3–4):279–292, 1992.

Linrui Zhang, Li Shen, Long Yang, Shixiang Chen, Bo Yuan, Xueqian Wang, and Dacheng Tao. Penalized proximal policy optimization for safe reinforcement learning. In *International Joint Conference on Artificial Intelligence*, 2022.

Jiayi Zhou. Pull request for omnisafe (feat: support discrete environment #286). `https://github.com/PKU-Alignment/omnisafe/pull/286`, November 2023.

A Algorithm Pseudocodes

Algorithm 1: **Safe Deep Q-Network using a proportional controller.**
(Changes from DQN (Mnih et al., 2015) are shown in black.)

initialize replay buffer D
initialize action-value function Q with random weights θ
initialize target action-value function Q_{prev} with weights $\theta_{\text{prev}} = \theta$
initialize Lagrangian parameters $\lambda = 0$ and episode cost $C_{\text{ep}} = 0$
initialize iteration count $t = 0$

for N episodes **do**
  get first state s
  calculate feature $\phi := \phi(s)$ // $\phi(s)$ can be the identity
  while s is not terminal **do**
    choose a using ε-greedy policy on $Q(\phi, \cdot)$
    take action a and observe r, $\boldsymbol{c}$, s'
    update episodic cost $C_{\text{ep}} \leftarrow C_{\text{ep}} + c$
    calculate feature $\phi' := \phi(s')$
    store transition $(\phi, a, r, \boldsymbol{c}, \phi')$ in D
    sample random mini batch $(\phi_b, a_b, r_b, \boldsymbol{c_b}, \phi'_b)$ from D
    if ϕ'_b terminal **then**
      $Q_{\text{target}} := r_b - \boldsymbol{\lambda} \cdot \boldsymbol{c_b}$
    else
      $Q_{\text{target}} := r_b - \boldsymbol{\lambda} \cdot \boldsymbol{c_b} + \gamma \max_{a'} Q_{\text{prev}}(\phi'_b, a')$
    end if
    calculate loss $L = \big(Q_{\text{target}} - Q(\phi_b, a_b)\big)^2$
    update Q-network using gradient descent $\theta \leftarrow \theta - \alpha \nabla_\theta L$ with learning rate α
    set $\phi \leftarrow \phi'$ and $\boldsymbol{t} \leftarrow \boldsymbol{t} + 1$
    every k_{sync} steps set $\theta_{\text{prev}} = \theta$
    anneal exploration variable ε
  end while
  if (s is terminal **or** truncated) **and** ($t > t_{\text{penalty}}$) **then**
    update $\lambda \leftarrow \max(0, \lambda + \alpha_\lambda (C_{\text{ep}} - b))$ with learning rate α_λ
    set $C_{\text{ep}} = 0$
  end if
end for

Algorithm 2: **Safe Deep Successor Representation using a proportional controller.**
(Changes from DSR (Kulkarni et al., 2016) are shown in black.)

initialize replay buffer D
initialize successor features M with random weights θ
initialize target successor features M_{prev} with weights $\theta_{\text{prev}} = \theta$
initialize feature network ϕ with random weights ξ
initialize inverse feature network g with random weights $\tilde{\xi}$
initialize reward weights w_r randomly
initialize cost weights w_c randomly
initialize target features ϕ_{prev} with weights $\xi_{\text{prev}} = \xi$
initialize Lagrangian parameters $\lambda = 0$ and episode cost $C_{\text{ep}} = 0$
initialize iteration count $t = 0$

for N episodes **do**
  get first state s
  while s is not terminal **do**
    choose a using ε-greedy policy on $Q(s, \cdot) - \lambda \cdot K(s, \cdot)$
    take action a and observe r, c, s'
    store transition (s, a, r, c, s') in D
    update episodic cost $C_{\text{ep}} \leftarrow C_{\text{ep}} + c$
    sample random mini batch (s_b, a_b, r_b, c_b) from D using biased sampling
    calculate loss $L_{\text{reward}} = \big(r_b - \phi(s_b, a_b) \cdot w_r\big)^2$
    calculate loss $L_{\text{cost}} = \big|\big|c_b - w_c \cdot \phi(s, a)\big|\big|_2^2$
    calculate loss $L_{\text{reconstruction}} = \big|\big|(s_b, a_b) - g(\phi(s_b, a_b))\big|\big|_2^2$
    calculate loss $L_{\text{features}} = \beta_r L_{\text{reward}} + \beta_c L_{\text{cost}} + \beta_g L_{\text{reconstruction}}$ with scaling factors $\beta_r, \beta_c, \beta_g$
    if $t > t_{\text{freeze}}$ **then**
      update w_r and w_c using gradient descent on L_{features} with learning rate α
    else
      update w_r, w_c, ϕ and g using gradient descent on L_{features} with learning rate α
    end if
    sample random mini batch (s_b, a_b, s'_b) from D using uniform sampling
    calculate next action $a'_b = \arg\max_{a'} \{Q(s'_b, a') - \lambda \cdot K(s'_b, a')\}$
    if s'_b terminal **then**
      $M_{\text{target}} := \phi_{\text{prev}}(s_b, a_b)$
    else
      $M_{\text{target}} := \phi_{\text{prev}}(s_b, a_b) + \gamma M_{\text{prev}}(\phi_{\text{prev}}(s'_b, a'_b))$
    end if
    calculate loss $L = \big|\big|M_{\text{target}} - M(\phi(s_b, a_b))\big|\big|_2^2$
    update M-network using gradient descent on L with learning rate α
    set $s \leftarrow s'$ and $t \leftarrow t + 1$
    every k_{sync} steps set $\theta_{\text{prev}} = \theta$ and $\xi_{\text{prev}} = \xi$
    anneal exploration variable ε
  end while
  if (s is terminal **or** truncated) **and** ($t > t_{\text{penalty}}$) **then**
    update $\lambda \leftarrow \max(0, \lambda + \alpha_\lambda(C_{\text{ep}} - b))$ with learning rate α_λ
    set $C_{\text{ep}} = 0$
  end if
end for

B Parameters

Table 1 contains the SafeDQN and Table 2 the SafeDSR parameters used during the experiments. The SafeDSR algorithm was tuned using the environment depicted in Figure 3 with a fixed Lagrangian parameter $\lambda = 0$, as extensive pre-tuning would hide the actual number of environment steps needed to find a safe solution. For both algorithms, we use the Adam optimizer (Kingma & Ba, 2014) with its default parameters for network training, except for the learning rates.

Table 1: **SafeDQN parameters used during the experiments.** We use the DQN parameters provided by Huang et al. (2022), except the ones used for the proportional controller and the ε-annealing, for which we use the same configurations as for SafeDSR.

Algorithm Parameters	**Value**
batch size	128
buffer size	10 000
discount factor γ	0.99
learning rate α	0.000 25
total time steps [environment steps]	500 000
training frequency [environment steps per training step]	10
training start time [environment steps]	10 000
target network frequency k_{sync} [environment steps per synchronization]	500
ε-Annealing	**Value**
end time [environment steps]	100 000
end value ε_∞	0.25
start time [environment steps]	20 000
start value ε_0	1
Lagrangian Parameter Tuning	**Value**
cost budget b	5
learning rate α_λ	0.001
start value λ_0	0
training start time t_{penalty} [environment steps]	100 000
Q-Network	**# Nodes**
dense layer	120
ReLU activation function	
dense layer	84
ReLU activation function	
dense layer	4

Table 2: **SafeDSR parameters used during the experiments.**

Algorithm Parameters	**Value**
batch size	256
buffer size	25 000
cost loss coefficient β_c	10
discount factor γ	0.99
feature freezing time t_{freeze} [environment steps]	50 000
learning rate α	0.001
reconstruction loss coefficient β_g	5
reward loss coefficient β_r	0.25
total time steps [environment steps]	500 000
training frequency [environment steps per training step]	10
training start time [environment steps]	15 000
target network frequency k_{sync} [environment steps per synchronization]	500
update steps [iterations per training step]	10
ε-Annealing	**Value**
end time [environment steps]	100 000
end value ε_∞	0.25
start time [environment steps]	20 000
start value ε_0	1
Lagrangian Parameter Tuning	**Value**
cost budget b	5
learning rate α_λ	0.001
start value λ_0	0
training start time t_{penalty} [environment steps]	100 000
Feature Network ϕ	**# Nodes**
dense layer	64
ReLU activation function	
dense layer	64
ReLU activation function	
dense layer	128
normalization layer	
Inverse Network g	**# Nodes**
dense layer	128
ReLU activation function	
dense layer	64
ReLU activation function	
dense layer	64
Successor Feature Network M	**# Nodes**
dense layer	128
ReLU activation function	
dense layer	128
ReLU activation function	
dense layer	128

C Algorithm Sketches

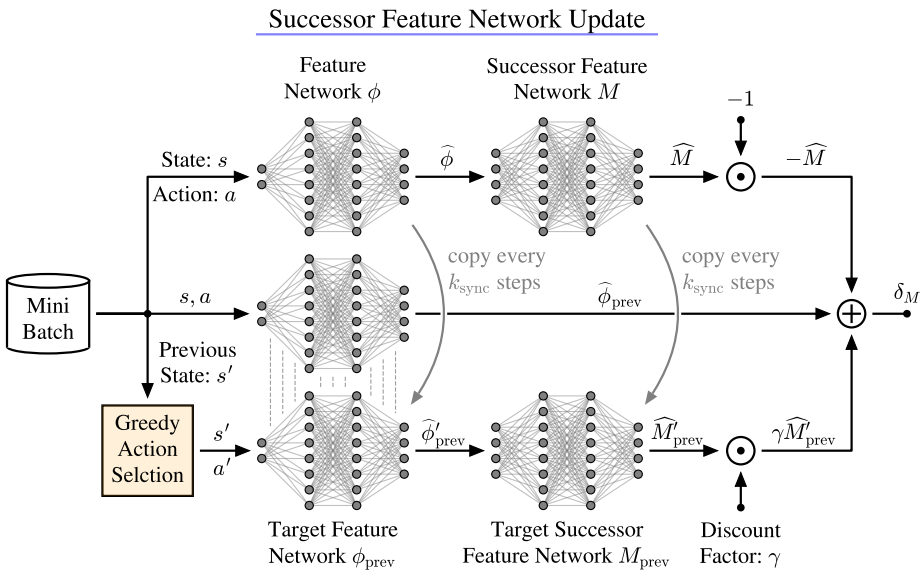

Figure 9: **Overview of the successor feature network update of the SafeDSR agent.** The agent samples mini batches (s, a, s') uniformly from the replay buffer. The upper branch calculates the future expected state occupancy $\widehat{M}$ for state–action pairs (s, a) in a high-dimensional space. The two lower branches estimate the target $\widehat{\phi}_{\text{prev}} + \gamma\widehat{M}'_{\text{prev}} := \phi_{\text{prev}}(s, a) + \gamma M_{\text{prev}}(\phi_{\text{prev}}(s', a'))$, where a' is the action the current network M would select in s'. The target networks M_{prev} and ϕ_{prev} are synchronized with the corresponding main networks every k_{sync} steps. The square of the temporal difference $\left|\left|\delta_M\right|\right|_2^2 = \left|\left|\widehat{\phi}_{\text{prev}} + \gamma\widehat{M}'_{\text{prev}} - \widehat{M}\right|\right|_2^2$ is used as the successor feature loss, which is optimized via gradient descent.

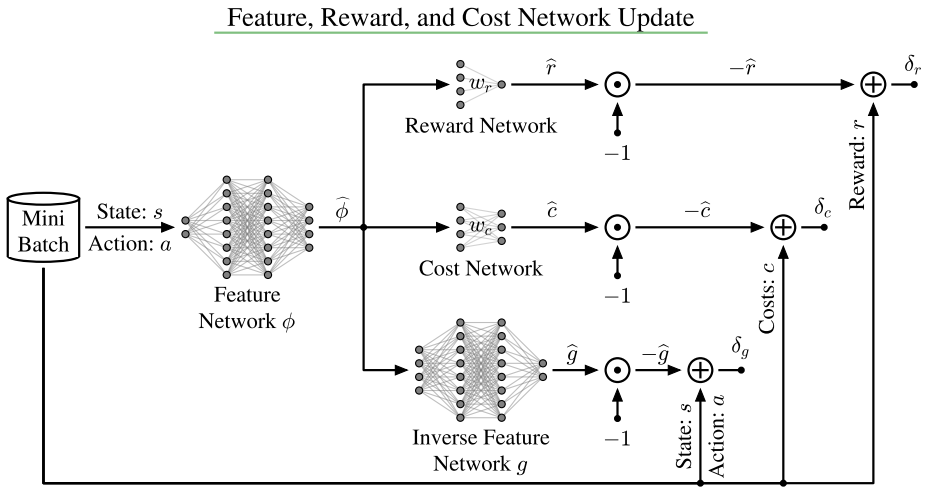

Figure 10: **Overview of the feature, reward, and cost network update of the SafeDSR agent.** The agent samples mini batches (s, a, r, c) from the replay buffer using biased sampling. The upper branch calculates the reward difference δ_r, the branch in the middle the cost difference δ_c, and the lower branch the reconstruction difference δ_g for the state-action pairs (s, a). All these differences are squared and summed up, yielding the feature loss $L_{\text{features}} = \delta_r^2 + ||\delta_c||_2^2 + ||\delta_g||_2^2$, which is optimized via gradient descent.

D Environment Examples

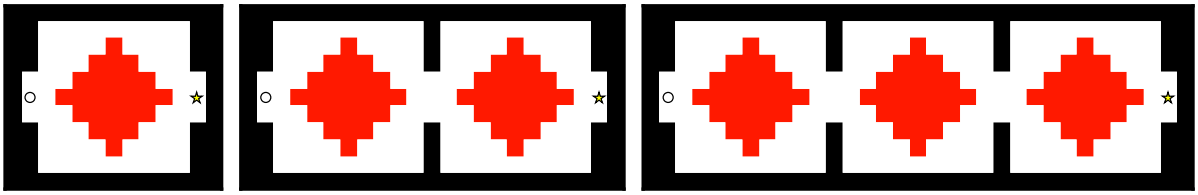

Figure 11: **Constrained grid environment with different number of rooms.** The exploration difficulty increases from left to right.

E Small Diamond to Large Diamond

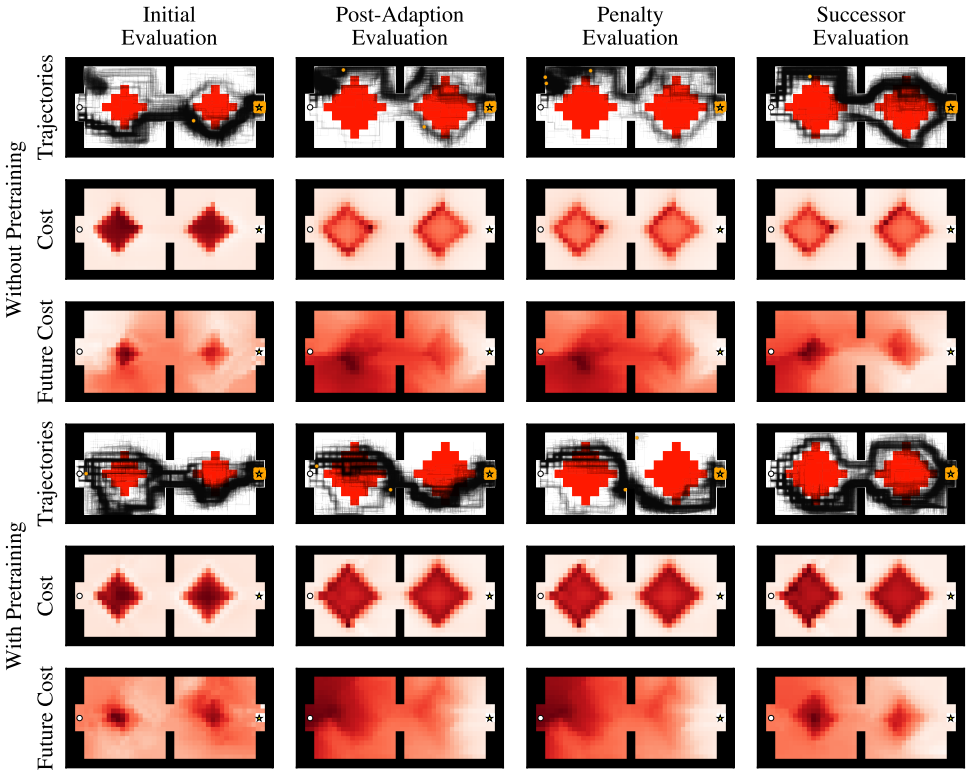

Figure 12: **Translation from a model pretrained on a smaller "diamond" shape towards a larger "diamond" distribution (Figure 3).** We refer to training reward and cost functions as "post-adaptation" evaluation, to training the Lagrangian on-policy as "penalty" evaluation, and finally to training the full network as "successor" evaluation. We also show the performance of our method with or without pretraining the feature embedding model.

F Average Safe Goal Rate When Changing Cost Distributions

We showcase in Figure 13 the change in safe goal rate as the costs get adapted. The numbers correspond to the different phases outlined in Table 3. It is worth noting that the absolute performance depends strongly on the difficulty of the constraints. To give an extreme example: if the red constraints block the entire path, then the maximally achievable safe goal rate is zero. Since the safe area changes in all evaluations (e.g., "square" only has a single feasible line at the top and bottom), one cannot directly compare safe goal rates.

For this experiment, we used a simpler Lagrangian parameter update rule than shown in Equation (16). We increased the Lagrangian parameter by 0.01 if the current episode violates the corresponding constraint, and otherwise decreases it by 0.01. This has the effect, that the average safe goal rate during the first phase in Figure 13 is smaller than in Figure 5 (middle). However, this change has no influence on the qualitative analysis in Section 5.4.

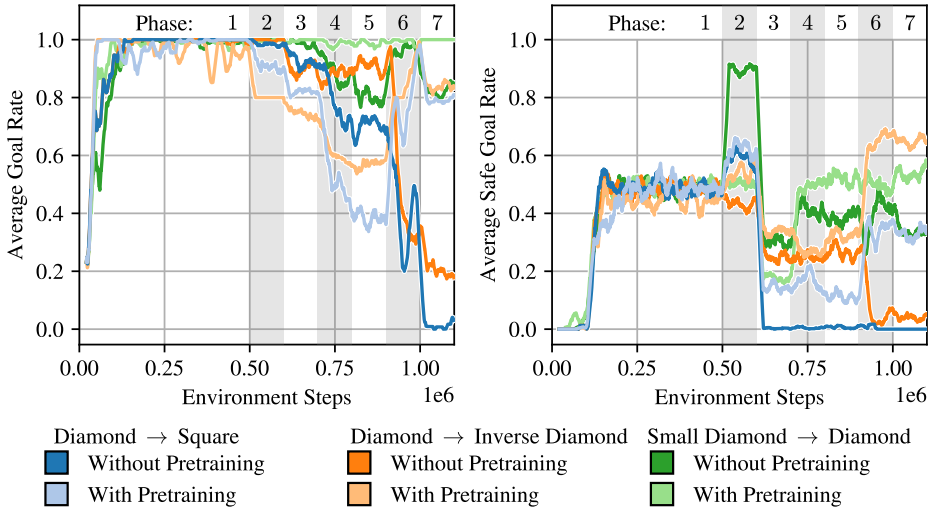

Figure 13: **Safe goal rate as the optimization cycles through different settings.** One has to be careful when comparing the different models against each other, as the achievable safe goal rate is highly dependent on the difficulty of the constraints.

Phase	Name	Start Time
1	initial training	0
2	initial evaluation	500 000
2.5	environment adaption	600 000
3	post-adaption evaluation	600 000
4	penalty learning	700 000
5	penalty evaluation	800 000
6	successor learning	900 000
7	successor evaluation	1 000 000

Table 3: **Different training and evaluation phases** of the experiment with changing cost distributions. We change the environment during the *environment adaption* phase, which lies between phase 2 and 3. The start time is given in environment steps and during evaluation we freeze all networks and parameters.