Exploration-Guided Reward Shaping for Reinforcement Learning under Sparse Rewards

Rati Devidze[1] rdevidze@mpi-sws.org
Parameswaran Kamalaruban[2] kparameswaran@turing.ac.uk
Adish Singla[1] adishs@mpi-sws.org

[1]Max Planck Institute for Software Systems (MPI-SWS), Saarbrucken, Germany
[2]The Alan Turing Institute, London, UK

Abstract

We study the problem of reward shaping to accelerate the training process of a reinforcement learning agent. Existing works have considered a number of different reward shaping formulations; however, they either require external domain knowledge or fail in environments with extremely sparse rewards. In this paper, we propose a novel framework, Exploration-Guided Reward Shaping (ExploRS), that operates in a fully self-supervised manner and can accelerate an agent's learning even in sparse-reward environments. The key idea of ExploRS is to learn an intrinsic reward function in combination with exploration-based bonuses to maximize the agent's utility w.r.t. extrinsic rewards. We theoretically showcase the usefulness of our reward shaping framework in a special family of MDPs. Experimental results on several environments with sparse/noisy reward signals demonstrate the effectiveness of ExploRS.

1 Introduction

Training reinforcement learning (RL) agents in environments with extremely sparse or distracting rewards is challenging. Existing works have studied several approaches to design informative rewards that speed up the agent's convergence [1–7]. One well-studied line of work is potential-based reward shaping, where a potential function is specified by an expert or obtained via transfer learning techniques (see [3, 8–17]). Another popular approach is to learn rewards via Inverse-RL using expert demonstrations [18]. Alternatively, one could also consider a manual specification of rewards, e.g., using distance-based metrics [19]. However, these reward design techniques typically rely on high-quality domain knowledge and may fail in practice. In fact, the RL agents can easily exploit poorly designed rewards and get stuck in local optima. This naturally leads to the fundamental question of how to do online reward shaping without relying on expert domain knowledge. More concretely, *can we design informative rewards that will accelerate the agent's training process by leveraging experience gained online during the agent's training lifetime itself?* [20–24]

To tackle this question, recent works [24–26] have explored fully self-supervised learning of parametric intrinsic rewards that can improve the performance of RL agents. In particular, these methods alternate between intrinsic reward parameter learning and the agent's policy optimization w.r.t. the learned reward. For instance, Learning Intrinsic Rewards for Policy Gradient (LIRPG) technique [25] updates the intrinsic reward parameters to maximize the extrinsic rewards received by the policy from the environment. Self-supervised Online Reward Shaping (SORS) technique [26] infers an intrinsic reward using a classification-based reward inference algorithm, TREX [27]. However, these fully self-supervised reward shaping techniques might fail to produce meaningful agent behavior in environments with extremely sparse rewards (called *hard-exploration* domains) as they lack an explicit explorative component. Intuitively, these techniques will not be able to make updates to parameters of their intrinsic reward functions, without receiving a non-zero extrinsic reward signal.

In a parallel line of work, several techniques have been proposed to specifically tackle the challenges of extreme sparsity and exploration. One such line of work is to add more stochasticity in the agent's behavior (e.g., [28–30]); however such techniques typically succeed in tasks with already well-shaped rewards. Another important line of work, relevant to our proposed framework, is bonus-driven exploration techniques for tackling hard-exploration domains – these techniques augment the extrinsic rewards with additional intrinsic bonus signals to encourage extra exploration [31]. A popular category of intrinsic bonuses is count-based bonuses that encourage RL agents to experience infrequently visited states [32–34]. Another category of intrinsic bonuses is providing rewards for improving the agent's knowledge about the environment [35–40]. However, simply relying on these bonus-driven signals can mislead the agent towards sub-optimal or bad behaviors — for instance, in *noisy-distractive* domains such as the "noisy TV" problem [41], unpredictable random or noisy outputs would attract the agent's attention forever.

An important research question that we seek to address is: *How can we design an online intrinsic reward function, without any domain knowledge, that can speed up the agent's learning process even in environments with extremely sparse rewards and noisy distractions?* To this end, we propose a novel framework, Exploration-Guided Reward Shaping (EXPLORS), that learns an intrinsic reward function in combination with exploration-based bonuses to maximize the agent's utility. EXPLORS operates in a fully self-supervised manner, and alternates between reward learning and policy optimization. Our main results and contributions are:

I. We propose a novel reward shaping framework, EXPLORS, that operates in a fully self-supervised manner and can accelerate an agent's learning even in sparse-reward environments. (Section 3.1).

II. We derive intuitive meta-gradients for updating the intrinsic reward component of EXPLORS that enables our framework to be broadly applicable to any RL agent and not only policy-gradient based agents (Sections 3.2 and 3.3).

III. We theoretically showcase the usefulness of our reward shaping framework in accelerating an agent's learning in a special family of chain environments (Section 3.4).

IV. We empirically demonstrate the effectiveness of EXPLORS on several environments with sparse and noisy reward signals (Section 4).[1]

2 Problem Setup

In Section 2.1, we present a general framework of online reward shaping technique for RL agents. In Section 2.2, we discuss the limitations of existing reward shaping techniques.

2.1 General Framework of Online Reward Shaping

Preliminaries. An environment is defined as a Markov Decision Process (MDP) $M := (\mathcal{S}, \mathcal{A}, T, P_0, \gamma, R)$, where the state and action spaces are denoted by $\mathcal{S}$ and $\mathcal{A}$ respectively. $T : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ captures the state transition dynamics, i.e., $T(s' \mid s, a)$ denotes the probability of landing in state s' by taking action a from state s. γ is the discounting factor, and P_0 is the initial state distribution. The reward function is given by $R : \mathcal{S} \times \mathcal{A} \rightarrow [-R_{\max}, R_{\max}]$, for some $R_{\max} > 0$. We denote the true underlying extrinsic reward function by $\overline{R}$ and the designed reward function by $\widehat{R}$. We denote a stochastic policy $\pi : \mathcal{S} \rightarrow \Delta(\mathcal{A})$ as a mapping from a state to a probability distribution over actions, and a deterministic policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ as a mapping from a state to an action. For any trajectory $\xi = \{(s_t, a_t)\}_{t=0,1,\ldots,H}$, we define its cumulative return w.r.t. reward function R as $J(\xi, R) := \sum_{t=0}^{H} \gamma^t \cdot R(s_t, a_t)$. Then, the expected cumulative return (value) of a policy π w.r.t. R is defined as $J(\pi, R) := \mathbb{E}\left[J(\xi, R) | P_0, T, \pi\right]$, where $s_0 \sim P_0(\cdot)$, $a_t \sim \pi(\cdot|s_t)$, and $s_{t+1} \sim T(\cdot|s_t, a_t)$. The learner seeks to find a policy that has maximum value w.r.t. the extrinsic reward function $\overline{R}$, i.e., $\max_\pi J(\pi, \overline{R})$.

[1]Github repo: https://github.com/machine-teaching-group/neurips2022_exploration-guided-reward-shaping.

Algorithm 1 Online Reward Shaping

```
1: Input: Extrinsic reward $\overline{R}$, and RL algorithm $L$
2: Initialization: $\pi_0$, $\widehat{R}_0$
3: for $k = 1, 2, \ldots, K$ do
4:     update policy $\pi_k \leftarrow L(\pi_{k-1}, \widehat{R}_{k-1})$
5:     update reward $\widehat{R}_k$ using $\widehat{R}_{k-1}$ and $\pi_k$
6: Output: $\pi_K$
```

Online reward shaping. A general framework of online reward shaping for RL agents is given in Algorithm 1. A natural objective here is to design informative rewards $\widehat{R}_k$ at each round k so that the resulting final policy π_K performs better (i.e., has high value w.r.t. $\overline{R}$) compared to the corresponding policy obtained via the standard training with $\widehat{R}_k = \overline{R}$. Note that we consider a single lifetime training setting for an RL agent on a single task, i.e., there is no resetting of the policy between rounds.

2.2 Existing Techniques and Issues

A popular technique for reward shaping is potential-based reward shaping (PBRS) which guarantees that any optimal policy induced by the designed reward function is also optimal under the extrinsic reward function [3]. However, for PBRS to be effective in accelerating the training process of an RL agent, we need to have access to good potential functions based on expert domain knowledge [42]. The focus of our work is on designing fully self-supervised reward shaping techniques. Below, we provide a discussion of existing techniques that do not require any expert guidance or domain knowledge, and also discuss their limitations.

Reward shaping based on exploration bonuses. In the bonus-driven exploration framework [32–34], a count-based intrinsic bonus $B_k(s)$ is given to the agent to encourage exploration. The bonus $B_k(s)$ measures the "novelty" of a state s given the history of all transitions up to round k. The authors in [34] extend the classic exploration methods with count-based intrinsic bonuses [43–46] to high-dimensional, continuous state spaces. However, these "exploration-only" reward shaping techniques do not appropriately combine the successful extrinsic reward signals received from the environment. When there are distractive zones in the state space, these methods will keep on exploring the state space even after obtaining extrinsic reward signals.

Fully self-supervised reward shaping: LIRPG [25]. Learning Intrinsic Rewards for Policy Gradient (LIRPG) technique [25] considers a parametric reward function of the form $\widehat{R}^{\text{LIRPG}}(s,a) = \overline{R}(s,a) + R_\phi(s,a)$, and learns the parameter ϕ of the intrinsic reward function R_ϕ in a fully self-supervised manner. LIRPG alternates between learning the intrinsic reward parameter ϕ and the agent's policy optimization w.r.t. the learned reward $\widehat{R}^{\text{LIRPG}}$. At round k, for fixed π_k, LIRPG updates the parameter ϕ_{k-1} to ϕ_k by considering the effect such a change would have on the expected cumulative return (w.r.t. $\overline{R}$) of the learner through the change in the policy π_k, i.e., update ϕ using the gradient $\left[\nabla_\phi J(L(\pi_k, \widehat{R}^{\text{LIRPG}}), \overline{R})\right]_{\phi_{k-1}}$. In order to develop an update rule for ϕ, LIRPG considers policy gradient style learning algorithm L with parametric policies $\left\{\pi_\theta : \theta \in \mathbb{R}^{d_\theta}\right\}$. More concretely, for a parameter θ_k at round k s.t. $\pi_k := \pi_{\theta_k}$, the learner's policy update depends on ϕ as $L(\pi_k, \widehat{R}^{\text{LIRPG}}) := \pi_{\theta(\phi)}$, where $\theta(\phi) = \theta_k + \alpha \cdot \left[\nabla_\theta J(\pi_\theta, \widehat{R}^{\text{LIRPG}})\right]_{\theta_k}$. Based on this learner update, the LIRPG update for the intrinsic reward parameters, at round k, is based on the following meta-gradients: $\phi_k = \phi_{k-1} + \eta \cdot [\nabla_\phi \theta(\phi)]_{\phi_{k-1}} \cdot \left[\nabla_{\theta(\phi)} J(\pi_{\theta(\phi)}, \overline{R})\right]_{\phi_{k-1}}$, where η is the learning rate. We note that the LIRPG technique could fail in environments with extremely sparse rewards as the agent may not receive a non-zero extrinsic reward signal needed to update the parameter ϕ. Moreover, the LIRPG technique is applicable only to policy-gradient based RL agents.

Fully self-supervised reward shaping: SORS [26]. Self-supervised Online Reward Shaping (SORS) technique [26] considers a reward function of the form $\widehat{R}^{\text{SORS}}(s,a) = R_\phi(s,a)$, and infers the parameter ϕ using a classification-based reward inference algorithm, T-REX [27]. However, unlike T-REX that requires rankings over the trajectories as input, SORS uses the extrinsic reward $\overline{R}$ as a self-supervised learning signal to rank the trajectories generated by the agent during training. By design, SORS only enforces the relative pairwise ordering over the trajectories w.r.t. $\overline{R}$ when training R_ϕ and ignores the scale of the returns associated with trajectories w.r.t. $\overline{R}$. This makes training a policy challenging when the environment has noisy or distractive reward signals. Further, similar to LIRPG, the SORS technique could fail in environments with extremely sparse rewards as the agent may not obtain any trajectories with non-zero extrinsic reward signal needed to update the parameter ϕ.

In this paper, we seek to develop an online reward shaping technique that can accelerate the agent's training process in environments with extremely sparse and distractive rewards, without any expert domain knowledge. As discussed above, techniques that rely only on intrinsic bonuses [32–34] could mislead the agent towards sub-optimal behaviors in noisy-distractive domains. Similarly, the fully self-supervised reward shaping techniques (LIRPG and SORS) might be ineffective in environments with extremely sparse rewards. We overcome these limitations by designing a novel reward shaping framework that appropriately balances exploration (via an intrinsic bonus component) and exploitation (via an intrinsic reward component) of extrinsic reward signals.

3 Exploration-Guided Reward Shaping

In Sections 3.1, 3.2, and 3.3, we propose an exploration-guided reward shaping framework, ExploRS, to accelerate an RL agent's training process. In Section 3.4, we theoretically showcase the usefulness of our framework in a chain environment.

3.1 Our Reward Formulation

We consider the following parametric reward function for ExploRS (see Algorithm 1):

$$\widehat{R}^{\textsc{ExploRS}}(s,a) \;:=\; \overline{R}(s,a) + R_\phi^{\textsc{SelfRS}}(s,a) + B_w^{\textsc{ExploB}}(s), \tag{1}$$

where $\phi \in \mathbb{R}^{d_\phi}$ and $w \in \mathbb{R}^{d_w}$. Here, $R_\phi^{\textsc{SelfRS}}$ corresponds to the intrinsic rewards in self-supervised reward shaping techniques, and $B_w^{\textsc{ExploB}}$ corresponds to the intrinsic bonuses in exploration-only reward shaping techniques. At round k of Algorithm 1, $\widehat{R}_{k-1}^{\textsc{ExploRS}}(s,a)$ is designed with parameters (ϕ_{k-1}, w_{k-1}). Then, given updated policy π_k, we update the parameters (ϕ_{k-1}, w_{k-1}) to (ϕ_k, w_k).

Notation. For the remainder of this section, we drop the superscripts (ExploRS, SelfRS, and ExploB) when referring to the reward functions in Eq. (1). In the subscript of the expectations $\mathbb{E}$, let $\pi(a|s)$ mean $a \sim \pi(\cdot|s)$, $\mu^\pi(s,a)$ mean $s \sim d^\pi, a \sim \pi(\cdot|s)$, and $\mu^\pi(s)$ mean $s \sim d^\pi$. Further, we use shorthand notation $\mu_{s,a}^k$ and μ_s^k to refer $\mu^{\pi_{\theta_k}}(s,a)$ and $\mu^{\pi_{\theta_k}}(s)$, respectively.

Intrinsic reward R_ϕ. We model the intrinsic reward R_ϕ using any parameterized function. At round k, for fixed π_k and w_{k-1}, we update the parameter ϕ_{k-1} to ϕ_k by considering the effect such a change would have on the the expected cumulative return w.r.t. $\overline{R}$ through the change in the policy π_k [24, 25]. In particular, we update ϕ using the gradient $\left[\nabla_\phi J(L(\pi_k, \widehat{R}), \overline{R})\right]_{\phi_{k-1}}$, where $\widehat{R}(s,a) = \overline{R}(s,a) + R_\phi(s,a) + B_{w_{k-1}}(s)$. However, when considering L with neural policies, it is challenging to directly analyze the impact of ϕ in the policy π_k. Since our goal is to design a reward shaping technique that is applicable to any RL agent, we consider a simple surrogate learning algorithm $\widetilde{L}$ for our analysis. In particular, we consider $\widetilde{L}$ with parametric policies $\left\{\pi_\theta : \theta \in \mathbb{R}^{d_\theta}\right\}$ that does single-step vanilla policy gradient update with Q-values computed using h-depth planning. We map the policy π_k to a parameter $\theta_k \in \mathbb{R}^{d_\theta}$ and define:

$$\widetilde{L}(\theta_k, \widehat{R}) := \; \theta_k + \alpha \cdot \left[\nabla_\theta J(\pi_\theta, \widehat{R})\right]_{\theta_k} \;=\; \theta_k + \alpha \cdot \mathbb{E}_{\mu_{s,a}^k}\left[\left[\nabla_\theta \log \pi_\theta(a|s)\right]_{\theta_k} Q_{\widehat{R},h}^{\pi_{\theta_k}}(s,a)\right],$$

where α is the learning rate and $Q_{\widehat{R},h}^{\pi_{\theta_k}}(s,a) = \mathbb{E}\left[\sum_{t=0}^{h} \gamma^t \widehat{R}(s_t,a_t) \middle| s_0 = s, a_0 = a, T, \pi_{\theta_k}\right]$ is the h-depth Q-value w.r.t. $\widehat{R}$. Then, we update ϕ using the following bi-level optimization:

$$\arg\max_{\phi} \quad J(\pi_{\theta(\phi)}, \overline{R}) \tag{P1.U}$$

$$\text{subject to} \quad \theta(\phi) \leftarrow \widetilde{L}(\theta_k, \widehat{R}), \tag{P1.L}$$

where $\widehat{R}(s,a) := \overline{R}(s,a) + R_\phi(s,a) + B_{w_{k-1}}(s)$. In the above bi-level formulation, $\widetilde{L}$ with h-depth planning for small values of h essentially requires designing more informative intrinsic rewards to benefit the agent's training process [24].

Intrinsic bonus B_w. Given a state abstraction $\psi : \mathcal{S} \to \mathcal{X}_\psi$ (with $|\mathcal{X}_\psi| = d_w$), we maintain the visitation count of the abstracted states in w, i.e., $w[x]$ corresponds to the visitation counts of the states $\{s \in \mathcal{S} : \psi(s) = x\}$. This allows us to implicitly maintain pseudo-counts $N_w(s)$ of visiting states $s \in \mathcal{S}$. In particular, we set $N_w(s) = \left(\frac{\lambda}{B_{\max}}\right)^2 + w[\psi(s)]$ for some $B_{\max}, \lambda > 0$. Then, we define the intrinsic bonus as follows: $B_w(s) = \frac{\lambda}{\sqrt{N_w(s)}}$. We update w based on the rollouts in round k [32–34].

3.2 Derivation of Gradient Updates for R_ϕ

In this subsection, we first obtain high-level meta-gradient updates for R_ϕ similar to LIRPG [25]. Then, we derive intuitive meta-gradient updates that would allow ExploRS to be compatible with any RL agent.

High-level gradient updates for R_ϕ. We solve the bi-level optimization problem (P1.U)-(P1.L) of the intrinsic reward component in an iterative manner using the gradient updates that we derive below. At round k, for fixed π_k and w_{k-1}, we update the parameter ϕ_{k-1} to ϕ_k as follows:

$$\begin{aligned}\phi_k \;=\; \phi_{k-1} + \eta \cdot \left[\nabla_\phi J(\pi_{\theta(\phi)}, \overline{R})\right]_{\phi_{k-1}} &\overset{(a)}{=} \phi_{k-1} + \eta \cdot \left[\nabla_\phi \theta(\phi) \cdot \nabla_{\theta(\phi)} J(\pi_{\theta(\phi)}, \overline{R})\right]_{\phi_{k-1}} \\ &\overset{(b)}{\approx} \phi_{k-1} + \eta \cdot \underbrace{\left[\nabla_\phi \theta(\phi)\right]_{\phi_{k-1}}}_{①} \cdot \underbrace{\left[\nabla_\theta J(\pi_\theta, \overline{R})\right]_{\theta_k}}_{②}, \qquad (2)\end{aligned}$$

where η is the learning rate, the equality in (a) is due to chain rule, and the approximation in (b) is made by assuming a smoothness condition of $\left\| \left[\nabla_\theta J(\pi_\theta, \overline{R})\right]_{\theta(\phi_{k-1})} - \left[\nabla_\theta J(\pi_\theta, \overline{R})\right]_{\theta_k} \right\|_2 \leq c \cdot \|\theta(\phi_{k-1}) - \theta_k\|_2$ for some $c > 0$. By using the meta-gradient derivations in [47–49], we write the term ① as follows: $\left[\nabla_\phi \theta(\phi)\right]_{\phi_{k-1}} = \alpha \cdot \mathbb{E}_{\mu^k_{s,a}}\left[\left[\nabla_\phi Q^{\pi_{\theta_k}}_{\widehat{R},h}(s,a)\right]_{\phi_{k-1}} \cdot \left[\nabla_\theta \log \pi_\theta(a|s)\right]^\top_{\theta_k}\right]$, where $\widehat{R}(s,a) := \overline{R}(s,a) + R_\phi(s,a) + B_{w_{k-1}}(s)$. By using the policy gradient theorem [50], we write the term ② as follows: $\left[\nabla_\theta J(\pi_\theta, \overline{R})\right]_{\theta_k} = \mathbb{E}_{\mu^k_{s,a}}\left[\left[\nabla_\theta \log \pi_\theta(a|s)\right]_{\theta_k} Q^{\pi_{\theta_k}}_{\overline{R}}(s,a)\right]$. The above gradient update of ϕ_k, involving the terms ① and ②, resembles the LIRPG [25] update. However, both the terms ① and ② require computing the gradient of the policy, i.e., $\nabla_\theta \log \pi_\theta(a|s)$. This requirement makes the above update applicable only for policy-gradient based agents. Below, we derive intuitive simplifications of the above two terms, ① and ②, that would enable our technique to be applicable to any RL agent, and not only policy-gradient based agents.

Intuitive gradient updates for R_ϕ. In order to obtain intuitive forms of the terms ① and ②, we consider further simplifications to the surrogate learning algorithm $\widetilde{L}$ introduced in Section 3.1. In particular, for our analysis and derivation, we let $\widetilde{L}$ use tabular representation $\theta \in \mathbb{R}^{|\mathcal{S}|\cdot|\mathcal{A}|}$ and softmax policy given by $\pi_\theta(a|s) := \frac{\exp\left(\theta(s,a)\right)}{\sum_b \exp\left(\theta(s,b)\right)}, \forall s \in \mathcal{S}, a \in \mathcal{A}$. We define $A^{\pi_{\theta_k}}_{\widehat{R},h}(s,a) := Q^{\pi_{\theta_k}}_{\widehat{R},h}(s,a) - V^{\pi_{\theta_k}}_{\widehat{R},h}(s)$ and $A^{\pi_{\theta_k}}_{\overline{R}}(s,a) := Q^{\pi_{\theta_k}}_{\overline{R}}(s,a) - V^{\pi_{\theta_k}}_{\overline{R}}(s)$. Based on this, the following proposition provides intuitive gradient updates for R_ϕ.

Proposition 1. *For the simplified surrogate learning algorithm $\widetilde{L}$ with h-depth planning, the gradient term $\left[\nabla_\phi \theta(\phi)\right]_{\phi_{k-1}} \cdot \left[\nabla_\theta J(\pi_\theta, \overline{R})\right]_{\theta_k}$ in Eq. (2) takes the following form:*

$$\alpha \cdot \mathbb{E}_{\mu^k_{s,a}}\left[\mu^k_{s,a} \cdot A^{\pi_{\theta_k}}_{\overline{R}}(s,a) \cdot \left[\nabla_\phi A^{\pi_{\theta_k}}_{\widehat{R},h}(s,a)\right]_{\phi_{k-1}}\right].$$

For the special case of $h = 1$, the gradient term further simplifies to the following form:

$$\alpha \cdot \mathbb{E}_{\mu^k_{s,a}}\left[\mu^k_{s,a} \cdot A^{\pi_{\theta_k}}_{\overline{R}}(s,a) \cdot \left[\nabla_\phi \big(R_\phi(s,a) - \mathbb{E}_{\pi_{\theta_k}(b|s)}[R_\phi(s,b)]\big)\right]_{\phi_{k-1}}\right].$$

Compared to Eq. (2), the intuitive gradient update term in the above proposition does not require computing the policy gradient $\nabla_\theta \log \pi_\theta(a|s)$. This allows us to develop an update rule for intrinsic reward parameter ϕ that is applicable to any RL agent. In particular, given the current policy π_k (possibly without any differentiable parameterization), we simplify Eq. (2) and propose the following gradient update rule for parameter ϕ:

$$\phi_k \approx \phi_{k-1} + \eta' \cdot \mathbb{E}_{\mu^k_{s,a}} \left[\mu^k_{s,a} \cdot A^{\pi_k}_{\overline{R}}(s,a) \cdot \left[\nabla_\phi \big(R_\phi(s,a) - \mathbb{E}_{\pi_k(b|s)}[R_\phi(s,b)] \big) \right]_{\phi_{k-1}} \right], \quad (3)$$

where $\eta' = \eta \cdot \alpha$. Note that the above gradient update only requires black-box access to the policy π_k in the form of trajectory rollouts as in the SORS technique [26].

3.3 Empirical Updates and Practical Aspects

In this subsection, we present a concrete pseudocode for training an RL agent with ExploRS reward shaping technique. Algorithm 2 provides a sketch of the overall training process, interleaving the agent's training with ExploRS. The sketch presented in Algorithm 2 is adapted from the training process proposed for the SORS technique [26]. Further, we consider rollouts where each round corresponds to a single rollout, instead of environment steps, as in SORS. Below, we discuss the empirical updates for intrinsic reward and bonus components of Eq. (1).

Empirical updates for intrinsic reward R_ϕ. We translate the final expectation-based update of ϕ_k in Eq. (3) to its empirical counterpart using the rollout data $\mathcal{D}$ collected by executing the current policy π_k (or recent policies) in the MDP M. At any round k, let $\mathcal{D}$ contain a collection of trajectories $\{\xi^i\}_{i=1}^n$, where $\xi^i = (s^i_0, a^i_0, s^i_1, a^i_1, \ldots, s^i_H)$. For a given trajectory ξ^i and time index t, we denote a partial trajectory as $\xi^i_t = (s^i_t, a^i_t, \ldots, s^i_H)$. Based on this notation, we empirically update the parameter ϕ as follows:

$$\phi_k \leftarrow \phi_{k-1} + \eta^\phi_k \cdot \sum_{\xi^i_t} \pi_k(a^i_t|s^i_t) \cdot \Big(J(\xi^i_t, \overline{R}) - V^{\pi_k}_{\overline{R}}(s^i_t) \Big) \cdot \left[\nabla_\phi A^{\pi_k}_{\widehat{R},1}(s^i_t, a^i_t) \right]_{\phi_{k-1}}, \quad (4)$$

where we absorb the normalization factors into η^ϕ_k, ignore the term $\mu^{\pi_k}(s^i_t)$, and set $\left[\nabla_\phi A^{\pi_k}_{\widehat{R},1}(s^i_t, a^i_t) \right]_{\phi_{k-1}} = \left[\nabla_\phi \left(R_\phi(s^i_t, a^i_t) - \mathbb{E}_{\pi_k(b|s^i_t)} \left[R_\phi(s^i_t, b) \right] \right) \right]_{\phi_{k-1}}$. Similar to LIRPG [25], we also maintain a critic $V_{\overline{R}, \widetilde{\phi}_{k-1}}(\cdot)$ to approximate $V^{\pi_k}_{\overline{R}}(\cdot)$ in Eq. (4). We update the parameters of the critic, $\widetilde{\phi}_{k-1}$ to $\widetilde{\phi}_k$, using the same rollout data $\mathcal{D}$ and learning rate $\eta^{\widetilde{\phi}}_k$. In Algorithm 2, hyper-parameters N_r and N_π control the frequency of updates for the intrinsic reward R_ϕ and policy π, respectively. For stability reasons, we update the policy more frequently compared to the intrinsic reward, i.e., $N_\pi < N_r$. We provide full implementation details in Section 4 and appendices.

Empirical updates for intrinsic bonus B_w. We update B_w based on the history of all the states visited up to round k. Similar to #Exploration [34], we use the count-based intrinsic bonuses with a state abstraction $\psi : \mathcal{S} \to \mathcal{X}_\psi$. We maintain the visitation count of the abstracted states in w. For each rollout ξ^k, we update the parameter w of the intrinsic bonus as follows:

$$w_k[x] \; = \; w_{k-1}[x] + \sum_{s^k_t \in \xi^k} \mathbf{1}\left\{ \psi(s^k_t) = x \right\}, \; \forall x \in \mathcal{X}_\psi. \quad (5)$$

Similar to the existing count-based exploration techniques [32–34], we use a lookahead step when incorporating the bonus term (see line 5 in Algorithm 2). In our implementation, we update the intrinsic bonus at a more fine-grained level, i.e., we update B_w at each environment step t within each round k directly, instead of waiting for the rollout to finish. However, for clear presentation in Algorithm 2, we write the B_w update at the level of round k, not at the level of environment step t. We provide full implementation details in Section 4 and appendices.

3.4 Theoretical Analysis

In this subsection, we theoretically showcase the usefulness of our exploration-guided reward shaping framework in accelerating an agent's learning in a chain environment with extremely-sparse rewards and distractive zones in the state space. Our analysis considers a stylized learning setting with simplified versions of different reward shaping techniques.

Algorithm 2 RL Training with ExploRS

```
1: Inputs and hyperparameters: RL algorithm $L$; first-in-first-out buffer $\mathcal{D}$ with size $D_{\text{max}}$;
   abstraction $\psi$; learning rates $\{\eta_k^{\phi}\}$, $\{\eta_k^{\widetilde{\phi}}\}$; bonus parameters $B_{\text{max}}$, $\lambda$; update rates $N_r$, $N_\pi$
2: Initialization: Initialize the parameters for intrinsic reward and its critic $(\phi_0, \widetilde{\phi}_0)$, parameters
   for intrinsic bonus $w_0$, and the policy $\pi_0$
3: for $k = 1, 2, \ldots, K$ do
       // policy update
4:     if $k\%N_\pi = 0$ then
5:         Define reward $\widehat{R}_{k-1}(s, a, s') := \overline{R}(s, a) + R_{\phi_{k-1}}(s, a) + B_{w_{k-1}}(s')$
6:         Obtain updated policy $\pi_k \leftarrow L(\pi_{k-1}, \widehat{R}_{k-1})$ using the latest rollouts in $\mathcal{D}$
7:     else
8:         Keep previous policy $\pi_k \leftarrow \pi_{k-1}$
       // data collection
9:     Rollout the policy $\pi_k$ in the MDP $M$ to obtain a trajectory $\xi^k = \left(s_0^k, a_0^k, s_1^k, a_1^k, \ldots, s_H^k\right)$
10:    Store $\xi^k$ in the buffer $\mathcal{D}$.add$(\xi^k)$; if the buffer $\mathcal{D}$ is full, remove the oldest trajectory
       // intrinsic reward update
11:    if $k\%N_r = 0$ then
12:        Obtain updated reward parameter $\phi_k$ from $\phi_{k-1}$ as in Eq. (4) using $\mathcal{D}$ and learning rate $\eta_k^{\phi}$
13:        Obtain updated critic parameter $\widetilde{\phi}_k$ from $\widetilde{\phi}_{k-1}$ using $\mathcal{D}$ and learning rate $\eta_k^{\widetilde{\phi}}$
14:    else
15:        Keep previous parameters $\phi_k \leftarrow \phi_{k-1}$ and $\widetilde{\phi}_k \leftarrow \widetilde{\phi}_{k-1}$
       // intrinsic bonus update
16:    Update $w_k$ as in Eq. (5) using the states visited in the trajectory $\xi^k$
17:    Define bonus $B_{w_k}(s) = \frac{\lambda}{\sqrt{N_{w_k}(s)}}$, where $N_{w_k}(s) = \left(\frac{\lambda}{B_{\text{max}}}\right)^2 + w_k[\psi(s)]$
18: Output: Policy $\pi_K$
```

Chain environment. We consider a chain environment $M = \left(\mathcal{S}, \mathcal{A}, T, P_0, \gamma, \overline{R}\right)$ of length $n_1 + n_2 + 1$. Let the state space be $\mathcal{S} = \{x_{-n_2}, \ldots, x_{-1}, x_0, x_1, \ldots, x_{n_1}\}$, and the action space be $\mathcal{A} = \{\leftarrow, \rightarrow\}$. We always start in the state x_0, i.e., the initial state distribution is $P_0(x_0) = 1$. The transition dynamics is deterministic and given as follows: $T(x_{i+1}|x_i, \rightarrow) = 1$ for $-n_2 \leq i \leq n_1 - 1$, $T(x_{i-1}|x_i, \leftarrow) = 1$ for $-(n_2 - 1) \leq i \leq n_1$, $T(\texttt{terminal}|x_{n_1}, \rightarrow) = 1$, and $T(\texttt{terminal}|x_{-n_2}, \leftarrow) = 1$. The reward function is defined as follows: $\overline{R}(x_i, \rightarrow) = 0$ for $-n_2 \leq i \leq n_1 - 1$, $\overline{R}(x_{n_1}, \rightarrow) = 1$, and $\overline{R}(x_i, \leftarrow) = 0$ for $-n_2 \leq i \leq n_1$. We consider an infinite horizon setting with discounted returns, i.e., $H \rightarrow \infty$ and $\gamma < 1$.

Learning algorithm and reward shaping techniques. For our theoretical analysis, we consider a stylized learning setting with a TD-style RL algorithm L and simplified versions of different reward shaping techniques; details are provided in appendices. We analyze the total number time steps required for L to learn an optimal policy in the chain environment under four different settings: (i) Case $L(\text{SelfRS} = 0, \text{ExploB} = 0)$ is a default setting without any shaping; (ii) Case $L(\text{SelfRS} = 0, \text{ExploB} = 1)$ uses only the intrinsic bonuses; (iii) Case $L(\text{SelfRS} = 1, \text{ExploB} = 0)$ uses only the intrinsic rewards; (iv) Case $L(\text{SelfRS} = 1, \text{ExploB} = 1)$ combines intrinsic bonuses with intrinsic rewards. The following theorem compares these four settings and showcases the usefulness of our framework, i.e., Case $L(\text{SelfRS} = 1, \text{ExploB} = 1)$ – proof is provided in appendices.

Theorem 1. *Consider the chain environment M and the algorithm L defined above. Let cost$(L(\text{SelfRS}, \text{ExploB}))$ denote the total number time steps required for $L(\text{SelfRS}, \text{ExploB})$ to learn an optimal policy in M. Then, we have the following (expected) costs for the four settings:*

(i) $\mathbb{E}\left[\textit{cost}(L(\text{SelfRS} = 0, \text{ExploB} = 0))\right] \geq 2^{n_1 - 1}$;

(ii) $\textit{cost}(L(\text{SelfRS} = 0, \text{ExploB} = 1)) = n_1 \cdot (n_1 + n_2 + 2)$;

(iii) $\mathbb{E}\left[\textit{cost}(L(\text{SelfRS} = 1, \text{ExploB} = 0))\right] \geq 2^{n_1 - 1}$;

(iv) $\textit{cost}(L(\text{SelfRS} = 1, \text{ExploB} = 1)) \leq n_1 + n_2 + 2$

The proof and additional details about the learning setting are provided in appendices.

4 Experimental Evaluation

In this section, we evaluate our reward shaping framework on three environments: CHAIN (Section 4.1), ROOM (Section 4.2), and LINEK (Section 4.3). CHAIN corresponds to a navigation task in a chain, adapted from the environment used for theoretical analysis in Section 3.4; this is a canonical environment used for studying extremely sparse-reward settings [7]. ROOM corresponds to a navigation task in a grid-world where the agent has to learn a policy to quickly reach the goal location in one of four rooms, starting from an initial location. Even though this environment has a small state/action space, it provides a very rich and intuitive problem setting to validate different reward shaping techniques. In fact, variants of ROOM have been used extensively in the literature [10, 11, 14, 17, 51–54]—the environment used in our experiments is adapted from [54]. LINEK corresponds to a navigation task in a one-dimensional space where the agent has to first pick the correct key and then reach the goal. The agent's location is represented as a point on a line segment. This environment is inspired by variants of navigation tasks in the literature where an agent needs to perform subtasks [3, 54, 55]—the environment used in our experiments is adapted from [54]. We give an overview of main results here, and provide a more detailed description of the setup and additional implementation details in appendices.

4.1 Evaluation on CHAIN

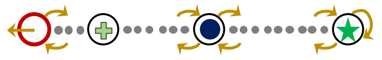


Figure 1: CHAIN^0 / CHAIN$^+$

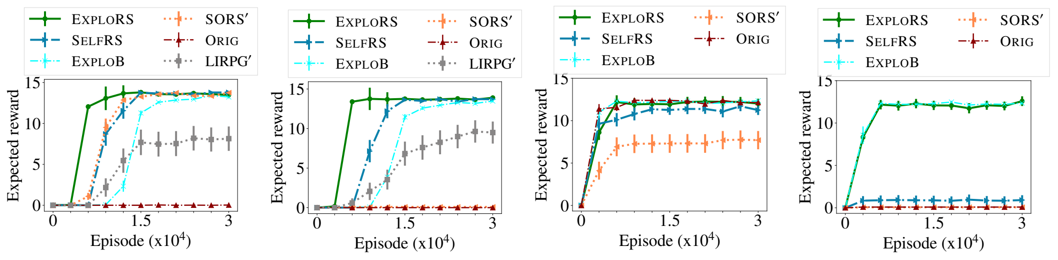
CHAIN (Figure 1). We represent the chain environment of length $n_1 + n_2 + 1$ as an MDP with state-space $\mathcal{S}$ consisting of an initial location x_0 (shown as "blue-circle"), n_1 nodes to the right of x_0, and n_2 nodes to the left of x_0. The rightmost node of the chain is the "goal" state (shown as "green-star"). In the left part of the chain, there can be a "distractor" state (shown as "green-plus"). The agent can take two actions given by $\mathcal{A} := \{\text{"left"}, \text{"right"}\}$ – an action takes the agent to the intended neighboring node with probability of $(1 - p_{\text{rand}})$ for $p_{\text{rand}} = 0.05$. The agent receives a reward of $R_{\max} = 1$ for the "right" action at the goal state, R_{dis} for the "left" action at the distractor state, and 0 for all other state-action pairs. There is a discount factor $\gamma = 0.99$ and the environment resets after a horizon of $H = n_2$ steps. We consider two different variants of the chain environment: (i) CHAIN^0 with $(n_1 = 20, n_2 = 40, R_{\text{dis}} = 0)$; (ii) CHAIN$^+$ with $(n_1 = 20, n_2 = 40, R_{\text{dis}} = 0.01)$. We defer the full environment details to appendices.

Evaluation setup. We conduct our experiments with two different types of RL agents for CHAIN: tabular REINFORCE agent [7] and tabular Q-learning agent [7]. Algorithm 2 provides a sketch of the overall training process, and shows how agent's training interleaves with reward shaping techniques. We compare the performance of the following reward shaping techniques: (i) $\widehat{R}^{\text{ORIG}} := \overline{R}$ is a default baseline without any shaping; (ii) $\widehat{R}^{\text{SORS'}} := \overline{R} + R_\phi^{\text{SORS}}$ is based on the SORS technique [26] (see Section 2.2);[2] (iii) $\widehat{R}^{\text{LIRPG'}}$ is obtained via adapting the LIRPG technique [25] to our training pipeline (see Algorithm 2, Sections 2.2 and 3.2)—note that $\widehat{R}^{\text{LIRPG'}}$ is not applicable to Q-learning agent;[3] (iv) $\widehat{R}^{\text{EXPLOB}} := \overline{R} + B_w^{\text{EXPLOB}}$ uses only the intrinsic bonuses; (v) $\widehat{R}^{\text{SELFRS}} := \overline{R} + R_\phi^{\text{SELFRS}}$ uses only the intrinsic rewards; (vi) $\widehat{R}^{\text{EXPLORS}} := \overline{R} + R_\phi^{\text{SELFRS}} + B_w^{\text{EXPLOB}}$ combines intrinsic bonuses with intrinsic rewards. We provide full details about the implementation and hyperparameters in appendices.

Results. During training, the agent receives rewards based on $\widehat{R}$ and is evaluated based on $\overline{R}$. Figure 2 shows results for both the variants of CHAIN environment; the reported results are averaged over 20 runs and convergence plots show the mean performance with standard error bars. These results demonstrate the effectiveness of our exploration-guided reward shaping framework ($\widehat{R}^{\text{EXPLORS}}$), in comparison to baselines ($\widehat{R}^{\text{ORIG}}$, $\widehat{R}^{\text{SORS'}}$, $\widehat{R}^{\text{LIRPG'}}$, $\widehat{R}^{\text{EXPLOB}}$, $\widehat{R}^{\text{SELFRS}}$). Next, we summarize some of our key findings. First, our results show that $\widehat{R}^{\text{EXPLORS}}$ outperforms the baselines in both CHAIN^0 and CHAIN$^+$ environments, irrespective of the RL agent (REINFORCE and Q-learning). Second, the performance of $\widehat{R}^{\text{EXPLORS}}$ is better than variants which only use either intrinsic bonuses or intrinsic

[2] In our implementation, we use a variant of the SORS technique which also incorporates the extrinsic reward component $\overline{R}$ as done in all other techniques in our evaluation setup.

[3] Throughout the experimental evaluation, we refer to our implementation of the LIRPG technique as $\widehat{R}^{\text{LIRPG'}}$ instead of $\widehat{R}^{\text{LIRPG}}$ – our implementation of the LIRPG technique is not based on computing meta-gradients as in the original work [25]. Instead, we implemented $\widehat{R}^{\text{LIRPG'}}$ as a variant of $\widehat{R}^{\text{SELFRS}}$ where we set $h \to \infty$ instead of 1 in $A^{\pi_{\theta_k}}_{\widehat{R},h}(s,a)$ (see Section 3.2). We provide additional implementation details in appendices.



(a) Chain^0, REINFORCE (b) Chain$^+$, REINFORCE (c) Chain^0, Q-learning (d) Chain$^+$, Q-learning

Figure 2: Results for Chain environment. These plots show convergence in performance of the agent w.r.t. training episodes. **(a, b)** show results for REINFORCE agent on Chain^0 (i.e., Chain variant without any distractor state) and Chain$^+$ (i.e., Chain variant with a distractor state). **(c, d)** show results for Q-learning agent on Chain^0 and Chain$^+$. See Section 4.1 for details.

rewards, i.e., $\widehat{R}^{\text{ExploB}}$ or $\widehat{R}^{\text{SelfRS}}$ – this demonstrates the utility of combining these two signals. Third, results in Figures 2b and 2d show that three reward shaping techniques ($\widehat{R}^{\text{SORS'}}$, $\widehat{R}^{\text{LIRPG'}}$, $\widehat{R}^{\text{SelfRS}}$) could fail or lead to sub-optimal policies because of the presence of distractor states.

4.2 Evaluation on Room

Room (Figure 3). This environment is based on the work of [54]; however, we adapted it to have a "distractor" state (shown as "green-plus") that provides a small reward. Similar to the two variants of Chain, we have two variants of this environment: (i) Room^0 has $R_{\text{dis}} = 0$ at the distractor state shown as "green-plus" (equivalently, there is no distractor state); (ii) Room$^+$ has $R_{\text{dis}} = 0.01$ at the distractor state. The environment-specific parameters (including p_{rand}, $R_{\max}$, γ) are kept same as in Section 4.1. We defer full details to appendices.

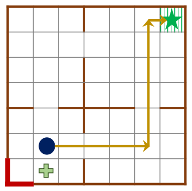


Figure 3: Room^0 / Room$^+$

Evaluation setup and results. Our evaluation setup for this environment is exactly same as that used for Chain environment (described in Section 4.1); here, we consider only the tabular REINFORCE agent. In particular, all the hyperparameters (related to the REINFORCE agent, reward shaping techniques, and training process) are the same as in Section 4.1. Figures 5a and 5b show the agent's performance for environments Room^0 and Room$^+$ (averaged over 20 runs). These results, along with results obtained in Figure 2, further demonstrate the effectiveness and robustness of $\widehat{R}^{\text{ExploRS}}$ across different environments in comparison to baselines.

4.3 Evaluation on LineK

LineK (Figure 4). This environment corresponds to a navigation task in a one-dimensional space where the agent has to first pick the correct key and then reach the goal. The environment used in our experiments is based on the work of [54]; however, we adapted it to have multiple keys (only one being correct) and "distractor" states that provide a small reward at goal locations even without the correct key. The environment comprises of the following main elements: (a) an agent whose current location (shown as "blue-circle") is a point x in $[0, 1]$; (b) goal (shown as "green-star") is available in locations on the segment $[0.9, 1]$; (c) a set of k keys that are available in locations on the segment $[0.0, 0.1]$, (d) among k keys, only 1 key is correct and the remaining $k - 1$ keys are wrong (i.e., irrelevant at the goal). Moreover, we consider the agent with two different actions related to picking a key: (a) "pickCorrect" makes the agent collect the correct key required at the goal; (b) "pickWrong" makes the agent collect one of the $k - 1$ wrong keys, chosen at random. Similar to Sections 4.1 and 4.2, we use two adaptations of the environment: (i) LineK^0 with $(k = 10, R_{\text{dis}} = 0)$; (ii) LineK$^+$ with $(k = 10, R_{\text{dis}} = 0.01)$. We defer full details to appendices.

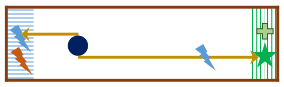
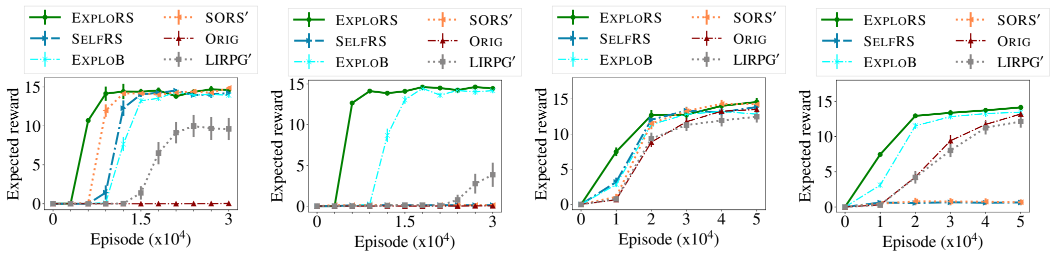


Figure 4: LineK^0 / LineK$^+$

Experimental setup. We conduct our experiments with a neural REINFORCE agent using a two-layered neural network architecture (i.e., one fully connected hidden layer with 256 nodes and RELU activation) [7]. Similar to Section 4.1, we compare the performance of six techniques. As a crucial difference, here we use neural-network based reward functions for $\widehat{R}^{\text{SORS'}}$, $\widehat{R}^{\text{LIRPG'}}$, $\widehat{R}^{\text{SelfRS}}$, and



(a) Room^0, REINFORCE (b) Room$^+$, REINFORCE (c) LineK^0, REINFORCE (d) LineK$^+$, REINFORCE

Figure 5: Results for Room and LineK environments. These plots show convergence in performance of the agent w.r.t. training episodes. **(a, b)** show results for REINFORCE agent on Room^0 (i.e., Room variant without any distractor state) and Room$^+$ (i.e., Room variant with a distractor state). **(c, d)** show results for REINFORCE agent on LineK^0 (i.e., LineK variant without any distractor state) and LineK$^+$ (i.e., LineK variant with distractor states). See Sections 4.2 and 4.3 for details.

$\widehat{R}^{\text{ExploRS}}$ (see Footnotes 2 and 3). Based on [25, 26], we use the same neural-network architecture for intrinsic reward functions as used for the agent's policy by applying appropriate transformations at the output layer (e.g., instead of using soft-max, use $tanh$-clipping to get output reward values for actions). We provide full details about the implementation and hyperparameters in appendices.

Results. During training, the agent receives rewards based on $\widehat{R}$ and is evaluated based on $\overline{R}$. Figures 5c and 5d show results for both the variants of LineK environment; the reported results are averaged over 30 runs and convergence plots show the mean performance with standard error bars. These plots showcase the performance of different techniques as we vary $R_{\text{dis}} \in \{0.00, 0.01\}$ – this in turn decides whether there are any distractor states that can serve as local minima for the agent. The convergence behavior in Figures 5c and 5d demonstrates the effectiveness of our exploration-guided reward shaping framework ($\widehat{R}^{\text{ExploRS}}$), in comparison to baselines ($\widehat{R}^{\text{Orig}}$, $\widehat{R}^{\text{SORS'}}$, $\widehat{R}^{\text{LIRPG'}}$, $\widehat{R}^{\text{ExploB}}$, $\widehat{R}^{\text{SelfRS}}$). Next, we summarize some of our key findings. First, our results show that $\widehat{R}^{\text{ExploRS}}$ outperforms all the baselines in both LineK^0 and LineK$^+$ environments. Second, results in Figure 5d show that three reward shaping techniques ($\widehat{R}^{\text{SORS'}}$, $\widehat{R}^{\text{LIRPG'}}$, $\widehat{R}^{\text{SelfRS}}$) performed worse than $\widehat{R}^{\text{Orig}}$ – this is because of the presence of distractor states which create local minima for the agent and these shaped functions could further encourage learning a sub-optimal policy. In contrast, $\widehat{R}^{\text{ExploRS}}$ combines the benefits of intrinsic rewards ($\widehat{R}^{\text{SelfRS}}$) and intrinsic bonuses ($\widehat{R}^{\text{ExploB}}$) to speed up agent's learning in a robust and efficient manner. Overall, these results demonstrate that our shaping technique $\widehat{R}^{\text{ExploRS}}$ results in efficient learning even when dealing with complex state representations and when learning neural-network based intrinsic reward functions.

5 Concluding Discussions

We proposed a novel reward shaping framework, ExploRS, that operates in a fully self-supervised manner and could accelerate an agent's learning even in sparse-reward environments. Next, we discuss a few limitations of our work and outline a future plan to address them. First, the experimental evaluation is conducted on simpler environments to study the performance of techniques w.r.t. the three characteristics of (a) hard exploration, (b) local minima, and (c) "noisy TV" problem. It would be interesting to evaluate different reward design techniques in more complex environments (e.g., with continuous state/action spaces); this would also require designing benchmark environments that systematically capture the above three characteristics. Second, ExploRS combines the intrinsic rewards and intrinsic bonuses that allows it to overcome the limitations of state-of-the-art techniques. It would be interesting to develop more principled ways to combine these two signals. Third, it would be useful to provide rigorous analysis of ExploRS in terms of convergence speed and stability of an agent.

6 Acknowledgments

Parameswaran Kamalaruban acknowledges support from The Alan Turing Institute. Funded/Co-funded by the European Union (ERC, TOPS, 101039090). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

References

[1] Maja J. Mataric. Reward Functions for Accelerated Learning. In *ICML*, 1994.

[2] Jette Randløv and Preben Alstrøm. Learning to Drive a Bicycle Using Reinforcement Learning and Shaping. In *ICML*, 1998.

[3] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. In *ICML*, 1999.

[4] Adam Laud and Gerald DeJong. The Influence of Reward on the Speed of Reinforcement Learning: An Analysis of Shaping. In *ICML*, 2003.

[5] Falcon Z. Dai and Matthew R. Walter. Maximum Expected Hitting Cost of a Markov Decision Process and Informativeness of Rewards. In *NeurIPS*, 2019.

[6] Jose A. Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, Johannes Brandstetter, and Sepp Hochreiter. RUDDER: Return Decomposition for Delayed Rewards. In *NeurIPS*, 2019.

[7] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018.

[8] Eric Wiewiora. Potential-Based Shaping and Q-Value Initialization are Equivalent. *Journal of Artificial Intelligence Research*, 19:205–208, 2003.

[9] Eric Wiewiora, Garrison W. Cottrell, and Charles Elkan. Principled Methods for Advising Reinforcement Learning Agents. In *ICML*, 2003.

[10] John Asmuth, Michael L. Littman, and Robert Zinkov. Potential-based Shaping in Model-based Reinforcement Learning. In *AAAI*, 2008.

[11] Marek Grzes and Daniel Kudenko. Plan-based Reward Shaping for Reinforcement Learning. In *International IEEE Conference on Intelligent Systems*, volume 2, pages 10–22, 2008.

[12] Sam Devlin and Daniel Kudenko. Dynamic Potential-based Reward Shaping. In *AAMAS*, 2012.

[13] Marek Grzes. Reward Shaping in Episodic Reinforcement Learning. In *AAMAS*, 2017.

[14] Alper Demir, Erkin Çilden, and Faruk Polat. Landmark Based Reward Shaping in Reinforcement Learning with Hidden States. In *AAMAS*, 2019.

[15] Prasoon Goyal, Scott Niekum, and Raymond J. Mooney. Using Natural Language for Reward Shaping in Reinforcement Learning. In *IJCAI*, 2019.

[16] Haosheng Zou, Tongzheng Ren, Dong Yan, Hang Su, and Jun Zhu. Reward Shaping via Meta-Learning. *CoRR*, abs/1901.09330, 2019.

[17] Yuqian Jiang, Suda Bharadwaj, Bo Wu, Rishi Shah, Ufuk Topcu, and Peter Stone. Temporal-Logic-Based Reward Shaping for Continuing Reinforcement Learning Tasks. In *AAAI*, 2021.

[18] Pieter Abbeel and Andrew Y Ng. Apprenticeship Learning via Inverse Reinforcement Learning. In *ICML*, 2004.

[19] Alexander Trott, Stephan Zheng, Caiming Xiong, and Richard Socher. Keeping Your Distance: Solving Sparse Reward Tasks Using Self-Balancing Shaped Rewards. In *NeurIPS*, 2019.

[20] Satinder P. Singh, Andrew G. Barto, and Nuttapong Chentanez. Intrinsically Motivated Reinforcement Learning. In *NeurIPS*, 2004.

[21] Satinder Singh, Richard L Lewis, and Andrew G Barto. Where Do Rewards Come From? In *CogSci*, 2009.

[22] Satinder Singh, Richard L Lewis, Andrew G Barto, and Jonathan Sorg. Intrinsically Motivated Reinforcement Learning: An Evolutionary Perspective. *IEEE Transactions on Autonomous Mental Development*, 2(2):70–82, 2010.

[23] Jonathan Sorg, Satinder P Singh, and Richard L Lewis. Internal Rewards Mitigate Agent Boundedness. In *ICML*, 2010.

[24] Jonathan Sorg, Satinder P. Singh, and Richard L. Lewis. Reward Design via Online Gradient Ascent. In *NeurIPS*, 2010.

[25] Zeyu Zheng, Junhyuk Oh, and Satinder Singh. On Learning Intrinsic Rewards for Policy Gradient Methods. In *NeurIPS*, 2018.

[26] Farzan Memarian, Wonjoon Goo, Rudolf Lioutikov, Scott Niekum, and Ufuk Topcu. Self-Supervised Online Reward Shaping in Sparse-Reward Environments. In *IROS*, 2021.

[27] Daniel Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. Extrapolating Beyond Suboptimal Demonstrations via Inverse Reinforcement Learning from Observations. In *ICML*, 2019.

[28] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-Level Control Through Deep Reinforcement Learning. *Nature*, 518(7540):529–533, 2015.

[29] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous Control with Deep Reinforcement Learning. *CoRR*, abs/1509.02971, 2015.

[30] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust Region Policy Optimization. In *ICML*, 2015.

[31] Lilian Weng. Exploration Strategies in Deep Reinforcement Learning. *lilianweng.github.io*, 2020.

[32] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying Count-Based Exploration and Intrinsic Motivation. In *NeurIPS*, 2016.

[33] Georg Ostrovski, Marc G Bellemare, Aäron Oord, and Rémi Munos. Count-Based Exploration with Neural Density Models. In *ICML*, 2017.

[34] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. #Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning. In *NeurIPS*, 2017.

[35] Jürgen Schmidhuber. Formal Theory of Creativity, Fun, and Intrinsic Motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, 2(3):230–247, 2010.

[36] Pierre-Yves Oudeyer, Frdric Kaplan, and Verena V Hafner. Intrinsic Motivation Systems for Autonomous Mental Development. *IEEE Transactions on Evolutionary Computation*, 11(2):265–286, 2007.

[37] Pierre-Yves Oudeyer and Frederic Kaplan. What is Intrinsic Motivation? A Typology of Computational Approaches. *Frontiers in Neurorobotics*, 1:6, 2009.

[38] Bradly C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing Exploration in Reinforcement Learning with Deep Predictive Models. *CoRR*, abs/1507.00814, 2015.

[39] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-Driven Exploration by Self-Supervised Prediction. In *ICML*, 2017.

[40] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational Information Maximizing Exploration. In *NeurIPS*, 2016.

[41] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by Random Network Distillation. *CoRR*, abs/1810.12894, 2018.

[42] Ching-An Cheng, Andrey Kolobov, and Adith Swaminathan. Heuristic-Guided Reinforcement Learning. In *NeurIPS*, 2021.

[43] Ronen I Brafman and Moshe Tennenholtz. R-max: A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning. *Journal of Machine Learning Research*, 3:213–231, 2002.

[44] Alexander L Strehl and Michael L Littman. An Analysis of Model-Based Interval Estimation for Markov Decision Processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008.

[45] J Zico Kolter and Andrew Y Ng. Near-Bayesian Exploration in Polynomial Time. In *ICML*, 2009.

[46] Jonathan Sorg, Satinder Singh, and Richard L Lewis. Variance-Based Rewards for Approximate Bayesian Reinforcement Learning. In *UAI*, 2010.

[47] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to Learn by Gradient Descent by Gradient Descent. In *NeurIPS*, 2016.

[48] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-Learning with Memory-Augmented Neural Networks. In *ICML*, 2016.

[49] Alex Nichol, Joshua Achiam, and John Schulman. On First-Order Meta-Learning Algorithms. *CoRR*, abs/1803.02999, 2018.

[50] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *NeurIPS*, 1999.

[51] Amy McGovern and Andrew G. Barto. Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density. In *ICML*, 2001.

[52] Özgür Simsek, Alicia P. Wolfe, and Andrew G. Barto. Identifying Useful Subgoals in Reinforcement Learning by Local Graph Partitioning. In *ICML*, 2005.

[53] Michael R. James and Satinder P. Singh. Sarsalandmark: An Algorithm for Learning in POMDPs with Landmarks. In *AAMAS*, 2009.

[54] Rati Devidze, Goran Radanovic, Parameswaran Kamalaruban, and Adish Singla. Explicable Reward Design for Reinforcement Learning Agents. In *NeurIPS*, 2021.

[55] Roberta Raileanu, Emily Denton, Arthur Szlam, and Rob Fergus. Modeling Others using Oneself in Multi-Agent Reinforcement Learning. In *ICML*, 2018.

Checklist

1. For all authors...
    (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] The paper is organized according to the contributions listed at the end of the introduction section.
    (b) Did you describe the limitations of your work? [Yes] In Section 5, we discuss the limitations of our work and outline a future plan to address these limitations.
    (c) Did you discuss any potential negative societal impacts of your work? [N/A] This work presents a new online reward shaping technique for reinforcement learning agents. Given the algorithmic and empirical nature of our work applied to learning agents, we do not foresee any direct negative societal impacts of our work in the present form.
    (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] We confirm that our paper conforms with the ethics review guidelines.
2. If you are including theoretical results...
    (a) Did you state the full set of assumptions of all theoretical results? [Yes] All assumptions are stated either as part of the theorem statements or the proofs.
    (b) Did you include complete proofs of all theoretical results? [Yes] Complete proofs of all theoretical results are included in appendices.
3. If you ran experiments...
    (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The code is provided as a URL in Footnote 1.
    (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Details are provided in appendices.
    (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Error bars are included in all the result graphs
    (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Details are provided in appendices.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
    (a) If your work uses existing assets, did you cite the creators? [N/A]
    (b) Did you mention the license of the assets? [N/A]
    (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
    (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
    (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
    (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
    (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
    (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Table of Contents

In this section, we give a brief description of the content provided in the appendices of the paper.

B Derivation of Gradient Updates for R_ϕ: Proof (Section 3.2)

Proof of Proposition 1. For any $s \in \mathcal{S}, a \in \mathcal{A}$, let $\mathbf{1}_{s,a} \in \mathbb{R}^{|\mathcal{S}|\cdot|\mathcal{A}|}$ denote a vector with 1 in the (s,a)-th entry and 0 elsewhere. First, we simplify the term ① as follows:

$$
\begin{aligned}
&\frac{1}{\alpha}\cdot\left[\nabla_\phi \theta(\phi)\right]_{\phi_{k-1}} \\
=\;& \mathbb{E}_{\mu^k_{s,a}}\left[\left[\nabla_\phi Q^{\pi_{\theta_k}}_{\widehat{R},h}(s,a)\right]_{\phi_{k-1}}\cdot\left[\nabla_\theta \log\pi_\theta(a|s)\right]^\top_{\theta_k}\right] \\
=\;& \mathbb{E}_{\mu^k_{s,a}}\left[\left[\nabla_\phi Q^{\pi_{\theta_k}}_{\widehat{R},h}(s,a)\right]_{\phi_{k-1}}\cdot\left(\mathbf{1}_{s,a}-\sum_{a'}\pi_{\theta_k}(a'|s)\cdot\mathbf{1}_{s,a'}\right)^\top\right] \\
=\;& \mathbb{E}_{\mu^k_{s}}\left[\sum_a \pi_{\theta_k}(a|s)\cdot\left[\nabla_\phi Q^{\pi_{\theta_k}}_{\widehat{R},h}(s,a)\right]_{\phi_{k-1}}\cdot\left(\mathbf{1}_{s,a}-\sum_{a'}\pi_{\theta_k}(a'|s)\cdot\mathbf{1}_{s,a'}\right)^\top\right] \\
=\;& \mathbb{E}_{\mu^k_{s}}\left[\sum_a \pi_{\theta_k}(a|s)\left[\nabla_\phi Q^{\pi_{\theta_k}}_{\widehat{R},h}(s,a)\right]_{\phi_{k-1}}\mathbf{1}^\top_{s,a}-\sum_a \pi_{\theta_k}(a|s)\left[\nabla_\phi Q^{\pi_{\theta_k}}_{\widehat{R},h}(s,a)\right]_{\phi_{k-1}}\left(\sum_{a'}\pi_{\theta_k}(a'|s)\mathbf{1}^\top_{s,a'}\right)\right] \\
=\;& \mathbb{E}_{\mu^k_{s}}\left[\sum_a \pi_{\theta_k}(a|s)\left[\nabla_\phi Q^{\pi_{\theta_k}}_{\widehat{R},h}(s,a)\right]_{\phi_{k-1}}\mathbf{1}^\top_{s,a}-\left[\nabla_\phi\sum_a \pi_{\theta_k}(a|s) Q^{\pi_{\theta_k}}_{\widehat{R},h}(s,a)\right]_{\phi_{k-1}}\left(\sum_{a'}\pi_{\theta_k}(a'|s)\mathbf{1}^\top_{s,a'}\right)\right] \\
=\;& \mathbb{E}_{\mu^k_{s}}\left[\sum_a \pi_{\theta_k}(a|s)\cdot\left[\nabla_\phi Q^{\pi_{\theta_k}}_{\widehat{R},h}(s,a)\right]_{\phi_{k-1}}\cdot\mathbf{1}^\top_{s,a}-\left[\nabla_\phi V^{\pi_{\theta_k}}_{\widehat{R},h}(s)\right]_{\phi_{k-1}}\cdot\left(\sum_{a'}\pi_{\theta_k}(a'|s)\cdot\mathbf{1}_{s,a'}\right)^\top\right] \\
=\;& \mathbb{E}_{\mu^k_{s}}\left[\sum_a \pi_{\theta_k}(a|s)\cdot\left[\nabla_\phi Q^{\pi_{\theta_k}}_{\widehat{R},h}(s,a)\right]_{\phi_{k-1}}\cdot\mathbf{1}^\top_{s,a}-\left[\nabla_\phi V^{\pi_{\theta_k}}_{\widehat{R},h}(s)\right]_{\phi_{k-1}}\cdot\left(\sum_{a}\pi_{\theta_k}(a|s)\cdot\mathbf{1}^\top_{s,a}\right)\right] \\
=\;& \mathbb{E}_{\mu^k_{s}}\left[\sum_a \pi_{\theta_k}(a|s)\cdot\left[\nabla_\phi Q^{\pi_{\theta_k}}_{\widehat{R},h}(s,a)\right]_{\phi_{k-1}}\cdot\mathbf{1}^\top_{s,a}-\sum_a \pi_{\theta_k}(a|s)\cdot\left[\nabla_\phi V^{\pi_{\theta_k}}_{\widehat{R},h}(s)\right]_{\phi_{k-1}}\cdot\mathbf{1}^\top_{s,a}\right] \\
=\;& \mathbb{E}_{\mu^k_{s,a}}\left[\left[\nabla_\phi Q^{\pi_{\theta_k}}_{\widehat{R},h}(s,a)\right]_{\phi_{k-1}}\cdot\mathbf{1}^\top_{s,a}-\left[\nabla_\phi V^{\pi_{\theta_k}}_{\widehat{R},h}(s)\right]_{\phi_{k-1}}\cdot\mathbf{1}^\top_{s,a}\right] \\
=\;& \mathbb{E}_{\mu^k_{s,a}}\left[\left[\nabla_\phi\left(Q^{\pi_{\theta_k}}_{\widehat{R},h}(s,a)-V^{\pi_{\theta_k}}_{\widehat{R},h}(s)\right)\right]_{\phi_{k-1}}\cdot\mathbf{1}^\top_{s,a}\right].
\end{aligned}
$$

Then, we simplify the term ② as follows:

$$
\begin{aligned}
&\left[\nabla_\theta J(\pi_\theta,\overline{R})\right]_{\theta_k} \\
=\;& \mathbb{E}_{\mu^k_{s,a}}\left[\left[\nabla_\theta\log\pi_\theta(a|s)\right]_{\theta_k}\cdot Q^{\pi_{\theta_k}}_{\overline{R}}(s,a)\right] \\
=\;& \mathbb{E}_{\mu^k_{s,a}}\left[\left(\mathbf{1}_{s,a}-\sum_{a'}\pi_{\theta_k}(a'|s)\cdot\mathbf{1}_{s,a'}\right)\cdot Q^{\pi_{\theta_k}}_{\overline{R}}(s,a)\right] \\
=\;& \mathbb{E}_{\mu^k_{s}}\left[\sum_a\pi_{\theta_k}(a|s)\cdot\left(\mathbf{1}_{s,a}-\sum_{a'}\pi_{\theta_k}(a'|s)\cdot\mathbf{1}_{s,a'}\right)\cdot Q^{\pi_{\theta_k}}_{\overline{R}}(s,a)\right]
\end{aligned}
$$

$$= \mathbb{E}_{\mu_s^k}\left[\sum_a \pi_{\theta_k}(a|s)\cdot Q_{\overline{R}}^{\pi_{\theta_k}}(s,a)\cdot \mathbf{1}_{s,a} - \sum_a \pi_{\theta_k}(a|s)\cdot Q_{\overline{R}}^{\pi_{\theta_k}}(s,a)\cdot\left(\sum_{a'}\pi_{\theta_k}(a'|s)\cdot \mathbf{1}_{s,a'}\right)\right]$$
$$= \mathbb{E}_{\mu_s^k}\left[\sum_a \pi_{\theta_k}(a|s)\cdot Q_{\overline{R}}^{\pi_{\theta_k}}(s,a)\cdot \mathbf{1}_{s,a} - V_{\overline{R}}^{\pi_{\theta_k}}(s)\cdot\left(\sum_{a'}\pi_{\theta_k}(a'|s)\cdot \mathbf{1}_{s,a'}\right)\right]$$
$$= \mathbb{E}_{\mu_s^k}\left[\sum_a \pi_{\theta_k}(a|s)\cdot Q_{\overline{R}}^{\pi_{\theta_k}}(s,a)\cdot \mathbf{1}_{s,a} - \sum_{a'}\pi_{\theta_k}(a'|s)\cdot V_{\overline{R}}^{\pi_{\theta_k}}(s)\cdot \mathbf{1}_{s,a'}\right]$$
$$= \mathbb{E}_{\mu_s^k}\left[\sum_a \pi_{\theta_k}(a|s)\cdot Q_{\overline{R}}^{\pi_{\theta_k}}(s,a)\cdot \mathbf{1}_{s,a} - \sum_{a}\pi_{\theta_k}(a|s)\cdot V_{\overline{R}}^{\pi_{\theta_k}}(s)\cdot \mathbf{1}_{s,a}\right]$$
$$= \mathbb{E}_{\mu_s^k}\left[\sum_a \pi_{\theta_k}(a|s)\cdot \left(Q_{\overline{R}}^{\pi_{\theta_k}}(s,a) - V_{\overline{R}}^{\pi_{\theta_k}}(s)\right)\cdot \mathbf{1}_{s,a}\right]$$
$$= \mathbb{E}_{\mu_{s,a}^k}\left[\left(Q_{\overline{R}}^{\pi_{\theta_k}}(s,a) - V_{\overline{R}}^{\pi_{\theta_k}}(s)\right)\cdot \mathbf{1}_{s,a}\right].$$

Finally, we consider the following:

$$\begin{aligned}
&\left[\nabla_\phi \theta(\phi)\right]_{\phi_{k-1}}\cdot\left[\nabla_\theta J(\pi_\theta,\overline{R})\right]_{\theta_k}\\
&= \alpha\cdot\mathbb{E}_{\mu_{s,a}^k}\left[\left[\nabla_\phi A_{\widehat{R},h}^{\pi_{\theta_k}}(s,a)\right]_{\phi_{k-1}}\cdot \mathbf{1}_{s,a}^\top\right]\cdot \mathbb{E}_{\mu_{s',a'}^k}\left[A_{\overline{R}}^{\pi_{\theta_k}}(s',a')\cdot \mathbf{1}_{s',a'}\right]\\
&= \alpha\cdot\mathbb{E}_{\mu_{s,a}^k}\left[\left[\nabla_\phi A_{\widehat{R},h}^{\pi_{\theta_k}}(s,a)\right]_{\phi_{k-1}}\cdot \mathbf{1}_{s,a}^\top\cdot \mathbb{E}_{\mu_{s',a'}^k}\left[A_{\overline{R}}^{\pi_{\theta_k}}(s',a')\cdot \mathbf{1}_{s',a'}\right]\right]\\
&= \alpha\cdot\mathbb{E}_{\mu_{s,a}^k}\left[\left[\nabla_\phi A_{\widehat{R},h}^{\pi_{\theta_k}}(s,a)\right]_{\phi_{k-1}}\cdot \mathbf{1}_{s,a}^\top\cdot \mu_{s,a}^k\cdot A_{\overline{R}}^{\pi_{\theta_k}}(s,a)\cdot \mathbf{1}_{s,a}\right]\\
&= \alpha\cdot\mathbb{E}_{\mu_{s,a}^k}\left[\mu_{s,a}^k\cdot A_{\overline{R}}^{\pi_{\theta_k}}(s,a)\cdot\left[\nabla_\phi A_{\widehat{R},h}^{\pi_{\theta_k}}(s,a)\right]_{\phi_{k-1}}\right]\\
&= \alpha\cdot\mathbb{E}_{\mu^{\pi_{\theta_k}}(s,a)}\left[\mu^{\pi_{\theta_k}}(s)\cdot\pi_{\theta_k}(a|s)\cdot A_{\overline{R}}^{\pi_{\theta_k}}(s,a)\cdot\left[\nabla_\phi A_{\widehat{R},h}^{\pi_{\theta_k}}(s,a)\right]_{\phi_{k-1}}\right].
\end{aligned}$$

□

C Theoretical Analysis: Proof (Section 3.4)

Proof of Theorem 1. We prove Theorem 1 via case-by-case analysis.

Case $L(\text{SELFRS}=0, \text{EXPLOB}=0)$. This case corresponds to learning without any reward shaping, i.e., learning with the extrinsic reward only: $\overline{R}(s,a)$. Then, we note the following:

I. Initially, we have a random policy except at state x_{n_1}, where we take the optimal action $\rightarrow$ (line 7). We maintain zero value function V_t for all the states (line 9) until we obtain the first success complete rollout, i.e., s_{t+1} is terminal and $\overline{R}(s_t,a_t)=1$.

II. With an initial random policy and starting from x_0, probability of obtaining a success complete rollout is $\left(\frac{1}{2}\right)^{n_1}+\left(\frac{1}{2}\right)^{n_1+2}+\left(\frac{1}{2}\right)^{n_1+4}+\ldots$, which is upper bounded by $p_{\text{max}} = \sum_{i=0}^{\infty}\left(\frac{1}{2}\right)^{n_1+i} = \left(\frac{1}{2}\right)^{n_1-1}$.

III. Let $\mathbb{E}\left[T_1\right]$ be the expected number of steps required for the first occurrence of the above successful rollout. Then, we have: $\mathbb{E}\left[T_1\right]\geq \frac{1}{p_{\text{max}}} = 2^{n_1-1}$.

IV. After the first successful rollout, we will have $V_t(x_{n_1})=1$ and zero elsewhere (line 9). Then, we will have a random policy except at x_{n_1} and $x_{(n_1-1)}$, where we take the optimal action (line 7). This effectively repeats the same steps above for the chain without x_{n_1}.

V. Let $\mathbb{E}\left[T_2\right]$ be the expected number of steps required for the second occurrence of the above successful rollout. Then, we have: $\mathbb{E}\left[T_2\right]\geq 2^{n_1-2}$.

Algorithm 3 Simplified RL Algorithm L with Reward Shaping

```
1: Input: Binary flags SELFRS and EXPLOB
2: Initialize: $V_0(s) = 0$; $R(s, a) = 0$, $B(s) = 1, \forall s \in \mathcal{S}, a \in \mathcal{A}$; $\lambda \in (0, 1)$
3: $s_1 = x_0$; $B(s_1) = \lambda$
4: for each $t = 1, 2, \ldots$ do
5:     if EXPLOB $= 0$ then
6:         $B(s) = 0, \forall s \in \mathcal{S}$
       // bonus component used for action selection
7:     $a_t = \arg\max_{a'} \overline{R}(s_t, a') + R(s_t, a') + B(T(s_t, a')) + \gamma \cdot V_{t-1}(T(s_t, a'))$
8:     $s_{t+1} = T(s_t, a_t)$
       // we do not consider the bonus component when updating the value function
9:     $V_t(s_t) = \overline{R}(s_t, a_t) + R(s_t, a_t) + \gamma \cdot V_{t-1}(s_{t+1})$
10:    if $s_{t+1} =$ terminal then
11:        if $\overline{R}(s_t, a_t) = 1$ and SELFRS $= 1$ then
               // update the intrinsic reward component
12:            $\phi(s) = 0, \forall s \in \mathcal{S}$
13:            Update $\phi(s)$ for all the states in the current rollout as the discounted return
14:            $R(s, a) = \gamma \cdot \phi(T(s, a)) - \phi(s), \forall s \in \mathcal{S}, a \in \mathcal{A}$
               // reset the value function to account for change in $R$
15:            $V_t(s) = 0, \forall s \in \mathcal{S}$
16:        reset $s_{t+1} = x_0$
       // update the bonus component
17:    $B(s_{t+1}) = \lambda \cdot B(s_{t+1})$
18: Output: policy $\pi_t$
```

VI. After the second successful rollout, we will have $V_t(x_{n_1}) = 1$, $V_t(x_{(n_1-1)}) = \gamma$, and zero elsewhere (line 9). Then, we will have a random policy except at x_{n_1}, $x_{(n_1-1)}$, and $x_{(n_1-2)}$, where we take the optimal action (line 7). This effectively repeats the same steps above for the chain without x_{n_1} and $x_{(n_1-1)}$.

VII. After following the above procedure for n_1 success rollouts, we will have the optimal value/policy learnt for the chain (solving the MDP). Thus, the expected sample complexity is lower bounded by $\mathbb{E}[\text{cost}(L(\text{SELFRS} = 0, \text{EXPLOB} = 0))] = \sum_{i=1}^{n_1} \mathbb{E}[T_i] \geq \sum_{i=1}^{n_1} 2^{n_1 - i}$.

Case $L(\text{SELFRS} = 0, \text{EXPLOB} = 1)$ This case corresponds to learning with the extrinsic reward and intrinsic bonus: $\overline{R}(s, a) + B(T(s, a))$. Then, we note the following (here, we need $\lambda \leq \gamma$):

I. We have zero value function (line 9) until we get the first success complete rollout, i.e., s_{t+1} is terminal and $\overline{R}(s_t, a_t) = 1$.

II. W.l.o.g. we take $\rightarrow$ action at time $t = 1$ at x_0. Then, we continue to take $\rightarrow$ action (for $n_1 + 1$ steps) until we reach rightmost terminal state, since $\lambda < 1$ (lines 7 and 17).

III. After the first successful rollout, we will have $V_t(x_{n_1}) = 1$ and zero elsewhere (line 9). Note that $V_t(\texttt{terminal}) = 0, \forall t$.

IV. Once we reset to x_0, we take $\leftarrow$ since $\lambda < 1$ (line 7). Then, we continue to take $\leftarrow$ action (for $n_2 + 1$ steps) until we reach leftmost terminal state, since $\lambda < 1$ (lines 7 and 17).

V. This alternating one-sided navigation process will continue until V_t values are updated for all the nodes right to x_0 (one node at a time per one full cycle). The condition $\lambda \leq \gamma$ ensures that after all the nodes right to x_0 get updated with right V_t values, there will be no further exploration on the left-side of x_0. Thus, the sample complexity is given by $\text{cost}(L(\text{SELFRS} = 0, \text{EXPLOB} = 1)) = n_1 \cdot (n_1 + n_2 + 2)$.

Case $L(\text{SELFRS} = 1, \text{EXPLOB} = 0)$ This case corresponds to learning with the extrinsic reward and intrinsic reward : $\overline{R}(s, a) + R(s, a)$. Then, we note the following:

I. From the analysis for the case $L(1,1)$, we have: $\mathbb{E}[T_1] \geq \frac{1}{p_{\max}} = 2^{n_1-1}$.

II. However, after the first successful rollout, we obtain the optimal policy (line 7) immediately since the shaping reward (line 14) contains myopic-optimality information. Thus, the expected sample complexity is lower bounded by $\mathbb{E}\left[\text{cost}(L(\textsc{SelfRS} = 1, \textsc{ExploB} = 0))\right] = \mathbb{E}[T_1] \geq 2^{n_1-1}$.

Case $L(\textsc{SelfRS} = 1, \textsc{ExploB} = 1)$ This case corresponds to learning with the extrinsic reward and intrinsic reward and bonus: $\overline{R}(s,a) + R(s,a) + B(T(s,a))$. Then, we note the following (here, we need $\lambda^2 \leq \gamma^{n_1}$):

I. From the analysis for the case $L(1,0)$, we obtain first successful trajectory after $n_1 + n_2 + 2$ steps (utmost). Then, as in the case of $L(0,1)$, shaping reward (line 14) will propagate myopic-optimality information immediately. The condition $\lambda^2 \leq \gamma^{n_1}$ ensures that after all the nodes right to x_0 get updated with right V_t values, there will be no further exploration on the left-side of x_0. Thus, the sample complexity is upper bounded by $\text{cost}(L(\textsc{SelfRS} = 1, \textsc{ExploB} = 1)) \leq n_1 + n_2 + 2$.

$\square$

D Evaluation on Chain: Additional Details (Section 4.1)

Chain (Figure 1). We expand on the details of the Chain environment, introduced in Section 4.1. We represent the chain environment of length $n_1 + n_2 + 1$ as an MDP with state-space $\mathcal{S}$ consisting of an initial location x_0 (shown as "blue-circle"), n_1 nodes to the right of x_0, and n_2 nodes to the left of x_0. The rightmost node of the chain is the "goal" state (shown as "green-star"). In the left part of the chain, there can be a "distractor" state (shown as "green-plus"). The agent can take two actions given by $\mathcal{A} := \{\text{"left"}, \text{"right"}\}$. An action takes the agent to the neighboring node represented by the direction of the action. However, taking "left" action at the leftmost node (shown as "thick-red-circle") leads to termination, and "right" action at the rightmost node (goal) keeps the agent at the current location. Furthermore, when an agent takes an action $a \in \mathcal{A}$, there is p_{rand} probability that an action $a' \in \mathcal{A} \setminus \{a\}$ will be executed instead of a. The agent receives rewards as follows: $R_{\max}$ for the "right" action at the goal state, R_{dis} for the "left" action at the distractor state, and 0 for all other state-action pairs. There is a discount factor γ and the environment resets after a horizon of $H = n_2$ steps. In our evaluation, we set $p_{\text{rand}} = 0.05$, $R_{\max} = 1$, $R_{\text{dis}} = 0$ or 0.01, and $\gamma = 0.99$. We obtain different variants of the chain environment by changing the values of $(n_1, n_2, R_{\text{dis}})$. We consider two different variants of the chain environment: (i) Chain^0 with $(n_1 = 20, n_2 = 40, R_{\text{dis}} = 0)$; (ii) Chain$^+$ with $(n_1 = 20, n_2 = 40, R_{\text{dis}} = 0.01)$. The "distractor" state (shown as "green-plus") with R_{dis} reward is located 15 nodes to the left of x_0 in both the environments.

Evaluation setup: agents. As mentioned in Section 4.1, we conduct our experiments with two different types of RL agents for Chain: tabular REINFORCE agent [7] and tabular Q-learning agent [7]. First, we consider tabular REINFORCE agent that maintain scores $\theta[s,a]$ for each state-action pair and applies soft-max operation over the scores to obtain the policy π. When computing the agent's performance during evaluation, we also use the agent's soft-max policy (instead of choosing actions greedily). Second, we consider tabular Q-learning agent with exploration factor $\epsilon = 0.05$. When computing the agent's performance during evaluation, we also use the agent's ϵ-greedy policy (instead of choosing actions greedily). Algorithm 2 provides a sketch of the overall training process, and shows how agent's training interleaves with reward shaping techniques – the agent's policy is updated in lines 4–8 of the algorithm. For the agent's training process, we use a fixed set of hyperparameters irrespective of the type of agent or the reward shaping technique. More concretely, we have the following: (a) the agent's learning rate is set to 0.1; (b) frequency of updates N_π is set to be 2, i.e., update after every 2 rollouts in the environment; (c) a rollout buffer (first-in-first-out) $\mathcal{D}$ of size 10 is maintained and we update the agent's policy using the last 5 rollouts in $\mathcal{D}$. In the tabular setting with Chain, we find that the overall quantitative results are robust to these hyperparameters – we use the exact same set of hyperparameters for evaluation on Room, described in Section 4.2.

Evaluation setup: shaping techniques. Next, we describe different reward shaping techniques used during the agent's training phase. Specifically, during training, the agent receives rewards based on the shaped reward function $\widehat{R}$; the performance (as reported in the plots) is always evaluated w.r.t. the extrinsic reward function $\overline{R}$. More concretely, we have the following shaping techniques:

- $\widehat{R}^{\text{ORIG}} := \overline{R}$. This serves as a default baseline where extrinsic reward function is used during training without any shaping.
- $\widehat{R}^{\text{SORS'}} := \overline{R} + R_{\phi}^{\text{SORS}}$. This is based on the SORS technique[26]; see additional details in Section 2.2 (also see Footnote 2 about $\widehat{R}^{\text{SORS'}}$). For CHAIN environment, we use tabular representation for R_{ϕ}^{SORS} and perform gradient updates as described in the work of [26]. Algorithm 2 provides a sketch of the overall training process – the R_{ϕ}^{SORS} updates would be applied in lines 11–15 in the algorithm. In fact, the training process presented in Algorithm 2 is adapted from the training process proposed for the SORS technique [26]. We update the intrinsic reward function using the following hyperparameters: (a) the learning rate is set to 0.01; (b) frequency of updates N_r is set to be 5, i.e., update after every 5 rollouts in the environment; (c) we have a rollout buffer $\mathcal{D}$ of size 10 and sample a set of 10 *pairs* of rollouts for the gradient updates (in our implementation, we prioritized sampling of pairs that have non-zero gap between returns).
- $\widehat{R}^{\text{LIRPG'}} := \overline{R} + R_{\phi}^{\text{LIRPG'}}$. This is obtained via adapting the LIRPG technique of [25] to our training pipeline; see Algorithm 2, Sections 2.2 and 3.2 (also see Footnote 3 about $\widehat{R}^{\text{LIRPG'}}$). More specifically, when considering tabular REINFORCE agent, we implemented $\widehat{R}^{\text{LIRPG'}}$ as an adaptation of $\widehat{R}^{\text{SELFRS}}$ where we set $h \to \infty$ instead of 1 (see Section 3.2) – the rest of the implementation is same as described below for $\widehat{R}^{\text{SELFRS}}$. Note that the LIRPG technique is not applicable to Q-learning agent.
- $\widehat{R}^{\text{EXPLOB}} := \overline{R} + B_{w}^{\text{EXPLOB}}$. This corresponds to a part of our reward shaping technique which uses only the intrinsic bonuses B_{w}^{EXPLOB}. As discussed in Sections 3.1 and 3.3, we use a count-based bonus B_{w}^{EXPLOB}. For CHAIN environment, we use a tabular representation for B_{w}^{EXPLOB} where $w[s]$ captures the state-visitation counts for a state s. Algorithm 2 provides a sketch of the overall training process – the B_{w}^{EXPLOB} updates are applied in lines 16–17 in the algorithm. We set the hyperparameters B_{max} and λ to be same as R_{max} ($= 1.0$ for CHAIN).[4]
- $\widehat{R}^{\text{SELFRS}} := \overline{R} + R_{\phi}^{\text{SELFRS}}$. This corresponds to a part of our reward shaping technique which uses only the intrinsic rewards R_{ϕ}^{SELFRS}. For CHAIN environment, we use a tabular representation for R_{ϕ}^{SELFRS} where $\phi[s,a]$ reward values are learned for each state-action pair and $R_{\phi}^{\text{SELFRS}}(s,a) := \phi[s,a]\ \forall (s,a)$. Along with R_{ϕ}^{SELFRS}, a tabular value-function $V_{\overline{R},\widetilde{\phi}}$ is maintained w.r.t. $\overline{R}$, serving as critic to compute values $V_{\overline{R}}^{\pi_k}(s)$ as needed for the empirical updates (see Section 3.3). For updating $V_{\overline{R},\widetilde{\phi}}$, we use Monte Carlo updates based on the trajectory returns as target and using a ℓ_2-norm loss function [7]. Algorithm 2 provides a sketch of the overall training process – the R_{ϕ}^{SELFRS} updates are applied in lines 11–15 in the algorithm. We set the following values for hyperparameters: (a) learning rate for updating ϕ parameters is set to 0.01; (b) learning rate for updating $\widetilde{\phi}$ parameters is set to 0.01; (c) frequency of updates N_r is set to be 5, i.e., update after every 5 rollouts in the environment; (d) we have a rollout buffer $\mathcal{D}$ of size 10. Furthermore, in all our experiments with Q-learning agent, we clipped the values of ϕ in the range $[-0.01, 0.01]$ (see Section 4.3 and Appendix F for another variant of clipping used with neural agents).
- $\widehat{R}^{\text{EXPLORS}} := \overline{R} + R_{\phi}^{\text{SELFRS}} + B_{w}^{\text{EXPLOB}}$. This is our exploration-guided reward shaping technique that combines intrinsic bonuses with intrinsic rewards. Algorithm 2 provides a sketch of the overall training process; we update R_{ϕ}^{SELFRS} and B_{w}^{EXPLOB} in the same way as described in the previous two points above.

Note that, for stability, we update the policy more frequently than the intrinsic reward ($N_{\pi} = 2$ vs. $N_r = 5$) and at a higher learning rate (0.1 vs. 0.01), as considered in the work of [25, 26]. In the tabular setting with CHAIN, we find that the overall quantitative results are robust to hyperparameters mentioned above – we use the exact same set of hyperparameters for evaluation on ROOM in Section 4.2.

[4]In our implementation, we do a more fine-grained update where the counts are updated during the rollout itself, instead of waiting for the end of the rollout. Moreoever, in our implementation, the bonus reward given for state-action (s,a) corresponds to bonus associated with the next state s' visited in the rollout.

Evaluation setup: compute resources. We ran the experiments on a cluster comprising of machines with 3.30 GHz Intel Xeon CPU E5-2667 v2 processor and 256 GB RAM.

E Evaluation on ROOM: Additional Details (Section 4.2)

ROOM (Figure 3). The environment used in our experiments is based on the work of [54]; however, we adapted it to have a "distractor" state (shown as "green-plus") that could provide a small positive reward. Next, we present additional details about the environment. We represent the environment as an MDP with $\mathcal{S}$ states, each corresponding to cells in the grid-world indicating the agent's current location (shown as "blue-circle"). The goal (shown as "green-star") is located at the top-right corner cell; in the bottom-left room, there can be a "distractor" state (shown as "green-plus") that could provide a small positive reward. The agent can take four actions given by $\mathcal{A} := \{$"up", "left", "down", "right"$\}$. An action takes the agent to the neighbouring cell represented by the direction of the action; however, if there is a wall (shown as "brown-segment"), the agent stays at the current location. There are also a few terminal walls (shown as "thick-red-segment") that terminate the episode, located at the bottom-left corner cell, where "left" and "down" actions terminate the episode. Furthermore, when an agent takes an action $a \in \mathcal{A}$, there is p_{rand} probability that an action $a' \in \mathcal{A} \setminus \{a\}$ will be executed instead of a. The agent gets a reward of $R_{\max}$ after it has navigated to the goal and then takes a "right" action (i.e., the reward can be accumulated in this state); similarly, the "up" action in the distractor state gives a reward of R_{dis}. The reward is 0 for all other state-action pairs. There is a discount factor γ and an episode terminates after $H = 30$ steps. The environment-specific parameters (including p_{rand}, $R_{\max}$, R_{dis}, γ) are kept same as in Section 4.1, i.e., $p_{\text{rand}} = 0.05$, $R_{\max} = 1$, $R_{\text{dis}} = 0$ or 0.01, and $\gamma = 0.99$. Similar to the two variants of CHAIN environment, we have two variants of this environment: (a) ROOM^0 has $R_{\text{dis}} = 0$ at the distractor state shown as "green-plus" (equivalently, there is no distractor state); (b) ROOM$^+$ has $R_{\text{dis}} = 0.01$ at the distractor state.

F Evaluation on LINEK: Additional Details (Section 4.3)

LINEK (Figure 4). We expand on the details of the LINEK environment, introduced in Section 4.3. As discussed in Section 4.3, this environment corresponds to a navigation task in a one-dimensional space where the agent has to first pick the correct key and then reach the goal. The environment used in our experiments is based on the work of [54]; however, we adapted it to have multiple keys (only one being correct) and "distractor" states that provide a small reward at goal locations even without the correct key. The environment comprises of the following main elements: (a) an agent whose current location (shown as "blue-circle") is a point x in $[0, 1]$; (b) goal (shown as "green-star") is available in locations on the segment $[0.9, 1]$; (c) a set of k keys that are available in locations on the segment $[0.0, 0.1]$; (d) among k keys, only 1 key is correct and the remaining $k - 1$ keys are wrong (i.e., irrelevant at the goal). The agent's initial location is sampled from $[0.3, 0.4]$.

The agent can take four actions given by $\mathcal{A} := \{$"left", "right", "pickCorrect", "pickWrong"$\}$. "pickCorrect" action does not change the agent's location, however, when executed in locations where keys are available, the agent acquires the correct key required at the goal; if the agent already possesses any key, the action has no effect. "pickWrong" action does not change the agent's location, however, when executed in locations where keys are available, the agent acquires one of the $k - 1$ wrong keys (chosen at random); if agent possesses a key, the action has no effect. A move action of type "left" or "right" takes the agent from the current location in the direction of the move with the dynamics of the final location captured by two hyperparameters $(\Delta_{a,1}, \Delta_{a,2})$; for instance, with current location x and action "left", the new location x$'$ is sampled uniformly among locations from $(\mathrm{x} - \Delta_{a,1} - \Delta_{a,2})$ to $(\mathrm{x} - \Delta_{a,1} + \Delta_{a,2})$. The agent's move action is not applied if the new location crosses the wall, and there is p_{rand} probability of a random action.

The agent receives rewards as follows: (a) $R_{\max}$ once it has navigated to the goal location after acquiring the correct key and then takes a "right" action (the action doesn't terminate the episode and reward can be accumulated); (b) R_{dis} after it has navigated to the goal location without acquiring the correct key and then takes a "right" action (the action doesn't terminate the episode and reward can be accumulated); (c) the reward is 0 elsewhere. We have a discount factor γ and the environment resets after a horizon of H. We set $p_{\text{rand}} = 0.05$, $R_{\max} = 1$, $R_{\text{dis}} = 0$ or 0.01, $H = 60$, $\gamma = 0.99$, $\Delta_{a,1} = 0.075$, and $\Delta_{a,2} = 0.01$.

We obtain different variants of the environment by changing the values of R_{dis} and number of keys k. Similar to Sections 4.1 and 4.2, we use two adaptations of the environment: (i) LINEK^0 with $(k = 10, R_{\text{dis}} = 0)$ (i.e., without any distractor state); (ii) LINEK$^+$ with $(k = 10, R_{\text{dis}} = 0.01)$ (i.e., with distractor states). In our experiments, we represent the environment as an MDP with $\mathcal{S}$ states comprising of the following: (a) the agent's current location (a point x in $[0, 1]$); (b) one bit indicating if the agent is on a segment with keys; (c) one bit indicating if the agent is on a segment with the goal; (d) k bits, corresponding to each of the k keys, indicating whether agent has that key or not (at most one of these bits can be one, as the agent can acquire only one key at any point in time, according to the transition dynamics specified above). This state representation is the input observation space for neural networks used by our policy and intrinsic reward functions.

Evaluation setup: agents. We conduct our experiments with a neural REINFORCE agent using a two-layered neural network architecture (i.e., one fully connected hidden layer with 256 nodes and RELU activation) [7]. In all the experiments that used neural-network based policies for agents, we also kept an exploration factor of $\epsilon = 0.05$, i.e., the agent uses soft-max neural policy with probability $(1 - \epsilon)$ and chooses a random action with ϵ. Algorithm 2 provides a sketch of the overall training process, and shows how agent's training interleaves with reward shaping techniques – the agent's policy is updated in lines 4–8 of the algorithm. For the agent's training process, we use a fixed set of hyperparameters irrespective of the type of reward shaping technique or specific variant of the environment. More concretely, we have the following: (a) the agent's learning rate is set to 10^{-5}; (b) frequency of updates N_π is set to be 2, i.e., update after every 2 rollouts in the environment; (c) a rollout buffer (first-in-first-out) $\mathcal{D}$ of size 10 is maintained and we update the agent's policy using the last 5 rollouts in $\mathcal{D}$. Most of these hyperparameters are close to what we used for the tabular REINFORCE agent in the CHAIN environment, described in Appendix D.

Evaluation setup: shaping techniques. Next, we describe different reward shaping techniques used during the agent's training phase. Specifically, during training, the agent receives rewards based on the shaped reward function $\widehat{R}$; the performance (as reported in the plots) is always evaluated w.r.t. the extrinsic reward function $\overline{R}$. Similar to Section 4.1, we compare the performance of six techniques. As a crucial difference, here we use neural-network based reward functions for $\widehat{R}^{\text{SORS'}}$, $\widehat{R}^{\text{LIRPG'}}$, $\widehat{R}^{\text{SELFRS}}$, and $\widehat{R}^{\text{EXPLORS}}$. We provide details of the different reward shaping techniques below:

- $\widehat{R}^{\text{ORIG}} := \overline{R}$. This serves as a default baseline where extrinsic reward function is used during training without any shaping.
- $\widehat{R}^{\text{SORS'}} := \overline{R} + R_\phi^{\text{SORS}}$. This is based on the SORS technique [26]; see additional details in Section 2.2 (also see Footnote 2 about $\widehat{R}^{\text{SORS'}}$). Following the neural architectures used for reward functions in [25, 26], we use the same neural-network architecture as used for the agent's policy – instead of using soft-max at the output layer to compute probability distribution over actions, here we use $tanh$-clipping (with a scaling factor of 0.10) to get output reward values for actions. Algorithm 2 provides a sketch of the overall training process – the R_ϕ^{SORS} updates would be applied in lines 11–15 in the algorithm. We update the intrinsic reward function using the following hyperparameters: (a) the learning rate is set to 10^{-3}; (b) frequency of updates N_r is set to be 20, i.e., update after every 20 rollouts in the environment; (c) we have a rollout buffer $\mathcal{D}$ of size 10 and sample a set of 10 *pairs* of rollouts for the gradient updates (in our implementation, we prioritized sampling of pairs that have non-zero gap between returns).
- $\widehat{R}^{\text{LIRPG'}} := \overline{R} + R_\phi^{\text{LIRPG'}}$. This is obtained via adapting the LIRPG technique of [25] to our training pipeline; see Algorithm 2, Sections 2.2 and 3.2 (also see Footnote 3 about $\widehat{R}^{\text{LIRPG'}}$). More specifically, in our experiments, we implemented $\widehat{R}^{\text{LIRPG'}}$ as an adaptation of $\widehat{R}^{\text{SELFRS}}$ where we set $h \to \infty$ instead of 1 in $A^{\pi_{\theta_k}}_{\widehat{R},h}(s, a)$ (see Section 3.2) – the rest of the implementation is same as described below for $\widehat{R}^{\text{SELFRS}}$. When computing $A^{\pi_{\theta_k}}_{\widehat{R},h}(s, a)$ for $h > 1$, we need an additional rollout to be able to compute this quantity. In our experiments with LINEK, we set $h \to \infty$ only for the starting state of the episode and kept $h = 1$ for the rest of the trajectory – this helped in reducing the computation time and variance.
- $\widehat{R}^{\text{EXPLOB}} := \overline{R} + B_w^{\text{EXPLOB}}$. This corresponds to a part of our reward shaping technique which uses only the intrinsic bonuses B_w^{EXPLOB}. As discussed in Sections 3.1 and 3.3, we use a count-based bonus B_w^{EXPLOB}. For this environment, we use an abstraction that discretizes the continuous

location part of the state to 0.1-length segments, i.e., creating 10 segments in total; the bits used to represent different indicator flags are then used along with these segments to represent an abstracted state. Given this abstraction, the rest of the process and hyperparameters for updating B_w^{ExploB} are the same as discussed in Appendix D.

- $\widehat{R}^{\text{SelfRS}} := \overline{R} + R_\phi^{\text{SelfRS}}$. This corresponds to a part of our reward shaping technique which uses only the intrinsic rewards R_ϕ^{SelfRS}. By following the neural architectures used for reward functions in [25, 26], we use the same neural-network architecture as used for the agent's policy. In particular, we use two networks for $\widehat{R}^{\text{SelfRS}}$: (a) one network is used for the reward function R_ϕ^{SelfRS} that applies $tanh$-clipping (with a scaling factor of 0.10) instead of soft-max to get output reward values for actions; (b) the second network is used for learning value-function $V_{\overline{R},\widetilde{\phi}}$ that applies a linear layer instead of a soft-max layer to obtain state-values. For updating $V_{\overline{R},\widetilde{\phi}}$, we use Monte Carlo updates based on the trajectory returns as target and using a ℓ_2-norm loss function [7]. Algorithm 2 provides a sketch of the overall training process – the R_ϕ^{SelfRS} updates are applied in lines 11–15 in the algorithm. We set the following values for hyperparameters: (a) learning rate for updating ϕ parameters is set to 10^{-3}; (b) learning rate for updating $\widetilde{\phi}$ parameters is set to $5 \cdot 10^{-3}$; (c) frequency of updates N_r is set to be 20, i.e., update after every 20 rollouts in the environment; (d) we have a rollout buffer $\mathcal{D}$ of size 10.
- $\widehat{R}^{\text{ExploRS}} := \overline{R} + R_\phi^{\text{SelfRS}} + B_w^{\text{ExploB}}$. This is our exploration-guided reward shaping technique that combines intrinsic bonuses with intrinsic rewards. Algorithm 2 provides a sketch of the overall training process; we update R_ϕ^{SelfRS} and B_w^{ExploB} in the same way as described in the previous two points above.

We update the policy more frequently than the intrinsic reward ($N_\pi = 2$ vs. $N_r = 20$), as considered in the work of [25, 26]. Moreover, for the first 5000 episodes of training, we do not supply intrinsic reward signals from neural network components of $\widehat{R}^{\text{SORS'}}$, $\widehat{R}^{\text{LIRPG'}}$, $\widehat{R}^{\text{SelfRS}}$, or $\widehat{R}^{\text{ExploRS}}$ (even though we keep updating their neural network components as usual) – this helps in preventing spuriourous reward signals associated with initialization of neural networks.