

---

# Learning to Utilize Shaping Rewards: A New Approach of Reward Shaping

---

**Yujing Hu<sup>1</sup>, Weixun Wang<sup>1,2</sup>, Hangtian Jia<sup>1</sup>, Yixiang Wang<sup>1,3</sup>, Yingfeng Chen<sup>1\*</sup>,  
Jianye Hao<sup>2,4</sup>, Feng Wu<sup>3</sup>, Changjie Fan<sup>1</sup>**

<sup>1</sup>Netease Fuxi AI Lab, Netease, Inc., Hangzhou, China

<sup>2</sup>College of Intelligence and Computing, Tianjin University, Tianjin, China

<sup>3</sup>School of Computer Science and Technology, University of Science and Technology of China

<sup>4</sup>Noah’s Ark Lab, Huawei, China

huyujing@corp.netease.com, wxwang@tju.edu.cn, jiahangtian@corp.netease.com  
yixiangw@mail.ustc.edu.cn, chenyfeng1@corp.netease.com  
jianye.hao@tju.edu.cn, wufeng02@ustc.edu.cn, fanchangjie@corp.netease.com

## Abstract

Reward shaping is an effective technique for incorporating domain knowledge into reinforcement learning (RL). Existing approaches such as potential-based reward shaping normally make full use of a given shaping reward function. However, since the transformation of human knowledge into numeric reward values is often imperfect due to reasons such as human cognitive bias, completely utilizing the shaping reward function may fail to improve the performance of RL algorithms. In this paper, we consider the problem of adaptively utilizing a given shaping reward function. We formulate the utilization of shaping rewards as a bi-level optimization problem, where the lower level is to optimize policy using the shaping rewards and the upper level is to optimize a parameterized shaping weight function for true reward maximization. We formally derive the gradient of the expected true reward with respect to the shaping weight function parameters and accordingly propose three learning algorithms based on different assumptions. Experiments in sparse-reward *cartpole* and *MuJoCo* environments show that our algorithms can fully exploit beneficial shaping rewards, and meanwhile ignore unbeneficial shaping rewards or even transform them into beneficial ones.

## 1 Introduction

A common way for addressing the sample efficiency issue of RL is to transform possible domain knowledge into additional rewards and guide learning algorithms to learn faster and better with the combination of the original and new rewards, which is known as reward shaping (RS). Early work of reward shaping can be dated back to the attempt of using hand-crafted reward function for robot behavior learning [3] and bicycle driving [15]. The most well-known work in the reward shaping domain is the potential-based reward shaping (PBRS) method [12], which is the first to show that policy invariance can be guaranteed if the shaping reward function is in the form of the difference of potential values. Recently, reward shaping has also been successfully applied in more complex problems such as Doom [8, 18] and Dota 2 [13].

Existing reward shaping approaches such as PBRS and its variants [2, 5, 6, 24] mainly focus on the way of generating shaping rewards (e.g., using potential values) and normally assume that the shaping rewards transformed from prior knowledge are completely helpful. However, such an assumption

---

\*Corresponding Author

is not practical since the transformation of human knowledge (e.g., rules) into numeric values (e.g., rewards or potentials) inevitably involves human operations, which are often subjective and may introduce human cognitive bias. For example, for training a Doom agent, designers should set appropriate rewards for key events such as object pickup, shooting, losing health, and losing ammo [8]. However, there are actually no instructions indicating what specific reward values are appropriate and the designers most probably have to try many versions of reward functions before getting a well-performed agent. Furthermore, the prior knowledge provided may also be unreliable if the reward designers are not experts of Doom.

Instead of studying how to generate useful shaping rewards, in this paper, we consider how to adaptively utilize a given shaping reward function. The term *adaptively utilize* stands for utilizing the beneficial part of the given shaping reward function as much as possible and meanwhile ignoring the unbeneficial shaping rewards. The main contributions of this paper are as follows. Firstly, we define the utilization of a shaping reward function as a bi-level optimization problem, where the lower level is to optimize policy for shaping rewards maximization and the upper level is to optimize a parameterized shaping weight function for maximizing the expected accumulative true reward. Secondly, we provide formal results for computing the gradient of the expected true reward with respect to the weight function parameters and propose three learning algorithms for solving the bi-level optimization problem. Lastly, extensive experiments are conducted in *cart-pole* and *MuJoCo* environments. The results show that our algorithms can identify the quality of given shaping rewards and adaptively make use of them. In some tests, our algorithms even transform harmful shaping rewards into beneficial ones and help to optimize the policy better and faster.

## 2 Background

In this paper, we consider the policy gradient framework [21] of reinforcement learning (RL) and adopt Markov decision process (MDP) as the mathematical model. Formally, an MDP is a tuple  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, r, p_0, \gamma \rangle$ , where  $\mathcal{S}$  is the state space,  $\mathcal{A}$  is the action space,  $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$  is the state transition function,  $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  is the (expected) reward function,  $p_0 : \mathcal{S} \rightarrow [0, 1]$  is the probability distribution of initial states, and  $\gamma \in [0, 1]$  is the discount rate. Generally, the policy of an agent in an MDP is a mapping  $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$  and can be represented by a parameterized function (e.g., a neural network). In this paper, we denote a policy by  $\pi_\theta$ , where  $\theta$  is the parameter of the policy function. Let  $p(s' \rightarrow s, t, \pi_\theta)$  be the probability that state  $s$  is visited after  $t$  steps from state  $s'$  under the policy  $\pi_\theta$ . We define the discounted state distribution  $\rho^\pi(s) = \int_{\mathcal{S}} \sum_{t=1}^{\infty} \gamma^{t-1} p_0(s') p(s' \rightarrow s, t, \pi_\theta) ds'$ , which is also called the discounted weighting of states when following. The goal of the agent is to optimize the parameter  $\theta$  for maximizing the expected accumulative rewards  $J(\pi_\theta) = \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta} [r(s, a)]$ . According to the policy gradient theorem [21], the gradient of  $J(\pi_\theta)$  with respect to  $\theta$  is  $\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) Q^\pi(s, a)]$ , where  $Q^\pi$  is the state-action value function.

### 2.1 Reward Shaping

Reward shaping refers to modifying the original reward function with a shaping reward function which incorporates domain knowledge. We consider the most general form, namely the additive form, of reward shaping. Formally, this can be defined as  $r' = r + F$ , where  $r$  is the original reward function,  $F$  is the shaping reward function, and  $r'$  is the modified reward function. Early work of reward shaping [3, 15] focuses on designing the shaping reward function  $F$ , but ignores that the shaping rewards may change the optimal policy. Potential-based reward shaping (PBRS) [12] is the first approach which guarantees the so-called policy invariance property. Specifically, PBRS defines  $F$  as the difference of potential values:  $F(s, a, s') = \gamma\Phi(s') - \Phi(s)$ , where  $\Phi : \mathcal{S} \rightarrow \mathbb{R}$  is a potential function which gives some kind of hints on states. Important variants of PBRS include the potential-based advice (PBA) approach:  $F(s, a, s', a') = \gamma\Phi(s', a') - \Phi(s, a)$ , which defines  $\Phi$  over the state-action space for providing advice on actions [24], the dynamic PBRS approach:  $F(s, t, s', t') = \gamma\Phi(s', t') - \Phi(s, t)$ , which introduces a time parameter into  $\Phi$  for allowing dynamic potentials [2], and the dynamic potential-based advice (DPBA) approach which learns an auxiliary value function for transforming any given rewards into potentials [6].

## 2.2 Related Work

Besides the shaping approaches mentioned above, other important works of reward shaping include the theoretical analysis of PBRS [23, 9], the automatic shaping approaches [11, 5], multi-agent reward shaping [1, 20], and some novel approaches such as belief reward shaping [10], ethics shaping [25], and reward shaping via meta learning [28]. Similar to our work, the automatic successive reinforcement learning (ASR) framework [4] learns to take advantage of multiple auxiliary shaping reward functions by optimizing the weight vector of the reward functions. However, ASR assumes that all shaping reward functions are helpful and requires the weight sum of these functions to be one. In contrast, our shaping approaches do not make such assumptions and the weights of the shaping rewards are state-wise (or state-action pair-wise) rather than function-wise. Our work is also similar to the optimal reward framework [17, 19, 27] which maximizes (extrinsic) reward by learning an intrinsic reward function and simultaneously optimizing policy using the learnt reward function. Recently, Zheng *et al.* [26] extend this framework to learn intrinsic reward function which can maximize lifetime returns and show that it is feasible to capture knowledge about long-term exploration and exploitation into a reward function. The most similar work to our paper may be the population-based method [7] which adopts a two-tier optimization process to learn the intrinsic reward signals of important game points and achieves human-level performance in the capture-the-flag game mode of Quake III. Learning an intrinsic reward function has proven to be an effective way for improving the performance of RL [14] and can be treated as a special type of online reward shaping. Instead of investigating how to learn helpful shaping rewards, our work studies a different problem where a shaping reward function is available, but how to utilize the function should be learnt.

## 3 Parameterized Reward Shaping

Given an MDP  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, r, p_0, \gamma \rangle$  and a shaping reward function  $f$ , our goal is to distinguish between the beneficial and unbeneficial rewards provided by  $f$  and utilize them differently when optimizing policy in  $\mathcal{M}$ . By introducing a *shaping weight function* into the additive form of reward shaping, the utilization problem of shaping rewards can be modeled as

$$\tilde{r}(s, a) = r(s, a) + z_\phi(s, a)f(s, a), \quad (1)$$

where  $z_\phi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  is the shaping weight function which assigns a weight to each state-action pair and is parameterized by  $\phi$ . In our setting, the shaping reward function is represented by  $f$  rather than  $F$  for keeping consistency with the other lower-case notations of rewards. We call this new form of reward shaping *parameterized reward shaping* as  $z_\phi$  is a parameterized function. For the problems with multiple shaping reward functions,  $z_\phi(s, a)$  is a weight vector where each element corresponds to one shaping reward function.

### 3.1 Bi-level Optimization

Let  $\pi_\theta$  denote an agent's (stochastic) policy with parameter  $\theta$ . The learning objective of parameterized reward shaping is twofold. Firstly, the policy  $\pi_\theta$  should be optimized according to the modified reward function  $\tilde{r}$ . Given the shaping reward function  $f$  and the shaping weight function  $z_\phi$ , this can be defined as  $\tilde{J}(\pi_\theta) = \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta} [r(s, a) + z_\phi(s, a)f(s, a)]$ . Secondly, the shaping weight function  $z_\phi$  needs to be optimized so that the policy  $\pi_\theta$ , which maximizes  $\tilde{J}(\pi_\theta)$ , can also maximize the expected accumulative true reward  $J(z_\phi) = \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta} [r(s, a)]$ . The idea behind  $J(z_\phi)$  is that although  $z_\phi$  cannot act as a policy, it can still be assessed by evaluating the true reward performance of its direct outcome (i.e., the policy  $\pi_\theta$ ). Thus, the optimization of the policy  $\pi_\theta$  and the shaping weight function  $z_\phi$  forms a bi-level optimization problem, which can be formally defined as

$$\begin{aligned} & \max_{\phi} \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta} [r(s, a)] \\ & \text{s.t. } \phi \in \Phi \\ & \theta = \arg \max_{\theta'} \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_{\theta'}} [r(s, a) + z_\phi(s, a)f(s, a)] \\ & \text{s.t. } \theta' \in \Theta, \end{aligned} \quad (2)$$

where  $\Phi$  and  $\Theta$  denote the parameter spaces of the shaping weight function and the policy, respectively. We call this problem *bi-level optimization of parameterized reward shaping* (BiPaRS, pronounced

“bypass”). In this paper, we use notations like  $x$  and  $\tilde{x}$  to denote the variables with respect to the original and modified MDPs, respectively. For example, we use  $r$  to denote the true reward function and  $\tilde{r}$  to denote the modified reward function.

### 3.2 Gradient Computation

The solution of the BiPaRS problem can be computed using a simple alternating optimization method. That is, to fix  $z_\phi$  or  $\pi_\theta$  and optimize the other alternately. Given the shaping weight function  $z_\phi$ , the lower level of BiPaRS is a standard policy optimization problem. Thus, the gradient of the expected accumulative modified reward  $\tilde{J}$  with respect to the policy parameter  $\theta$  is

$$\nabla_\theta \tilde{J}(\pi_\theta) = \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) \tilde{Q}(s, a)]. \quad (3)$$

Here,  $\tilde{Q}$  is the state-action value function of the current policy in the modified MDP  $\tilde{\mathcal{M}} = \langle \mathcal{S}, \mathcal{A}, P, \tilde{r}, p_0, \gamma \rangle$ . For the upper-level optimization, the following theorem is the basis for computing the gradient of  $J(z_\phi)$  with respect to the variable  $\phi$ .

**Theorem 1.** *Given the shaping weight function  $z_\phi$  and the stochastic policy  $\pi_\theta$  of the agent in the upper level of the BiPaRS problem (Equation (2)), the gradient of the objective function  $J(z_\phi)$  with respect to the variable  $\phi$  is*

$$\nabla_\phi J(z_\phi) = \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta} [\nabla_\phi \log \pi_\theta(s, a) Q^\pi(s, a)], \quad (4)$$

where  $Q^\pi$  is the state-action value function of  $\pi_\theta$  in the original MDP.

The complete proof is given in Appendix. Note that the theorem is based on the assumption that the gradient of  $\pi_\theta$  with  $\phi$  exists, which is reasonable because in the upper-level optimization, the given policy  $\pi_\theta$  is the direct result of applying  $z_\phi$  to reward shaping and in turn  $\pi_\theta$  can be treated as an implicit function of  $z_\phi$ . However, even with this theorem, we still cannot get the accurate value of  $\nabla_\phi J(z_\phi)$  because the gradient of the policy  $\pi_\theta$  with respect to  $\phi$  cannot be directly computed. In the next section, we will show how to approximate  $\nabla_\phi \pi_\theta(s, a)$ .

## 4 Gradient Approximation

### 4.1 Explicit Mapping

The idea of our first method for approximating  $\nabla_\phi \pi_\theta(s, a)$  is to establish an explicit mapping from the shaping weight function  $z_\phi$  to the policy  $\pi_\theta$ . Specifically, we redefine the agent’s policy as a hyper policy  $\pi_\theta : \mathcal{S}_z \times \mathcal{A} \rightarrow [0, 1]$  which can directly react to different shaping weights. Here,  $\mathcal{S}_z = \{(s, z_\phi(s)) | \forall s \in \mathcal{S}\}$  is the extended state space and  $z_\phi(s)$  is the shaping weight or shaping weight vector (e.g.,  $z_\phi(s) = (z_\phi(s, a_1), \dots, z_\phi(s, a_{|\mathcal{A}|}))$ ) corresponding to state  $s$ . As  $z_\phi$  is an input of the policy  $\pi_\theta$ , according to the chain rule we have  $\nabla_\phi \pi_\theta(s, a, z_\phi(s)) = \nabla_z \pi_\theta(s, a, z_\phi(s)) \nabla_\phi z_\phi(s)$  and correspondingly the gradient of the upper-level objective function  $J(z_\phi)$  with respect to  $\phi$  is

$$\nabla_\phi J(z_\phi) = \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta} [\nabla_z \log \pi_\theta(s, a, z)|_{z=z_\phi(s)} \nabla_\phi z_\phi(s) Q^\pi(s, a)]. \quad (5)$$

We call the first gradient approximating method *explicit mapping* (EM). Now we explain why the extension of the input space of  $\pi_\theta$  is reasonable. For the lower-level optimization, having  $z_\phi$  as the input of  $\pi_\theta$  redefines the modified MDP  $\tilde{\mathcal{M}} = \langle \mathcal{S}, \mathcal{A}, P, \tilde{r}, p_0, \gamma \rangle$  as a new MDP  $\tilde{\mathcal{M}}_z = \langle \mathcal{S}_z, \mathcal{A}, \tilde{P}_z, \tilde{r}_z, p_z, \gamma \rangle$ , where  $\tilde{r}_z$ ,  $\tilde{P}_z$  and  $p_z$  also include  $z_\phi$  in the input space and provide the same rewards, state transitions, and initial state probabilities as  $\tilde{r}$ ,  $P$ , and  $p_0$ , respectively. Given the shaping weight function  $z_\phi$ , the one-to-one correspondence between the states in  $\mathcal{S}$  and  $\mathcal{S}_z$  indicates that the two MDPs  $\tilde{\mathcal{M}}$  and  $\tilde{\mathcal{M}}_z$  are equivalent.

### 4.2 Meta-Gradient Learning

Essentially, the policy  $\pi_\theta$  and the shaping weight function  $z_\phi$  are related because that their parameters  $\theta$  and  $\phi$  are related. The idea of our second method is to approximate the meta-gradient  $\nabla_\phi \theta$  so that  $\nabla_\phi \pi_\theta(s, a)$  can be computed as  $\nabla_\theta \pi_\theta(s, a) \nabla_\phi \theta$ . Given  $\phi$ , let  $\theta$  and  $\theta'$  denote the policy parameters before and after one round of low-level optimization, respectively. Without loss of generality, we

assume that a batch of  $N$  ( $N > 0$ ) samples  $\mathcal{B} = \{(s_i, a_i) | i = 1, \dots, N\}$  is used for update  $\theta$ . According to Equation (3), we have

$$\theta' = \theta + \alpha \sum_{i=1}^N \nabla_\theta \log \pi_\theta(s_i, a_i) \tilde{Q}(s_i, a_i), \quad (6)$$

where  $\alpha$  is the learning rate. The updated policy  $\pi_{\theta'}$  then will be used for updating  $\phi$  in the subsequent round of upper-level optimization, which involves the computation of  $\nabla_\phi \theta'$ . For any state-action pair  $(s, a)$ , we simplify the notation  $\nabla_\theta \log \pi_\theta(s, a)$  as  $g_\theta(s, a)$ . Thus, the gradient of  $\theta'$  with respect to  $\phi$  can be computed as

$$\nabla_\phi \theta' = \nabla_\phi \left( \theta + \alpha \sum_{i=1}^N g_\theta(s_i, a_i) \tilde{Q}(s_i, a_i) \right) = \alpha \sum_{i=1}^N g_\theta(s_i, a_i)^\top \nabla_\phi \tilde{Q}(s_i, a_i). \quad (7)$$

We treat  $\theta$  as a constant with respect to  $\phi$  here because  $\theta$  is the result of the last round of low-level optimization rather than the result of applying  $z_\phi$  in this round. Now the problem is to compute  $\nabla_\phi \tilde{Q}(s_i, a_i)$ . Note that the state-action value function  $\tilde{Q}$  can be replaced by any of its unbiased estimations such as the Monte Carlo return. For any sample  $i$  in the batch  $\mathcal{B}$ , let  $\tau_i = (s_i^0, a_i^0, \tilde{r}_i^0, s_i^1, a_i^1, \tilde{r}_i^1, \dots)$  denote the sampled trajectory starting from  $(s_i, a_i)$ , where  $(s_i^0, a_i^0)$  is  $(s_i, a_i)$ . The modified reward  $\tilde{r}_i^t$  at each step  $t$  of  $\tau_i$  can be further denoted as  $\tilde{r}_i^t = r_i^t + z_\phi(s_i^t, a_i^t)f(s_i^t, a_i^t)$ , where  $r_i^t$  is the sampled true reward. Therefore,  $\nabla_\phi \theta'$  can be approximated as

$$\begin{aligned} \nabla_\phi \theta' &\approx \alpha \sum_{i=1}^N g_\theta(s_i, a_i)^\top \nabla_\phi \sum_{t=0}^{|\tau_i|-1} \gamma^t (r_i^t + z_\phi(s_i^t, a_i^t)f(s_i^t, a_i^t)) \\ &= \alpha \sum_{i=1}^N g_\theta(s_i, a_i)^\top \sum_{t=0}^{|\tau_i|-1} \gamma^t f(s_i^t, a_i^t) \nabla_\phi z_\phi(s_i^t, a_i^t). \end{aligned} \quad (8)$$

### 4.3 Incremental Meta-Gradient Learning

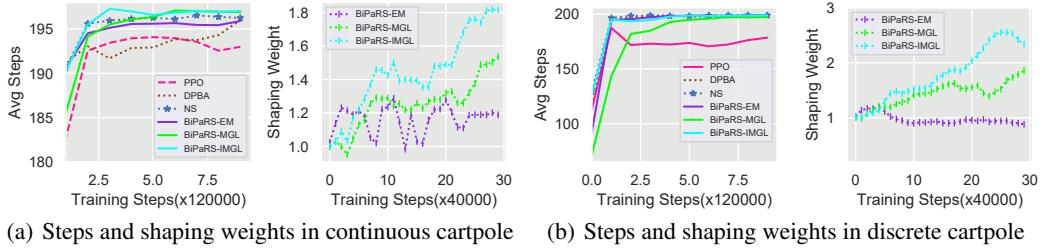
The third method for gradient approximation is a generalized version of the meta-gradient learning (MGL) method. Recall that in Equation (7), the old policy parameter  $\theta$  is treated as a constant with respect to  $\phi$ . However,  $\theta$  can be also considered as a non-constant with respect to  $\phi$  because in the last round of upper-level optimization,  $\phi$  is updated according to the true rewards sampled by the old policy  $\pi_\theta$ . Furthermore, as the parameters of the policy and the shaping weight function are updated incrementally round by round, both of them actually are related to all previous versions of the other. Therefore, we can rewrite Equation (7) as

$$\begin{aligned} \nabla_\phi \theta' &= \nabla_\phi \theta + \alpha \sum_{i=1}^N \nabla_\phi g_\theta(s_i, a_i)^\top \tilde{Q}(s_i, a_i) + \alpha \sum_{i=1}^N g_\theta(s_i, a_i)^\top \nabla_\phi \tilde{Q}(s_i, a_i) \\ &= \nabla_\phi \theta + \alpha \sum_{i=1}^N (\nabla_\theta g_\theta(s_i, a_i)^\top \nabla_\phi \theta) \tilde{Q}(s_i, a_i) + \alpha \sum_{i=1}^N g_\theta(s_i, a_i)^\top \nabla_\phi \tilde{Q}(s_i, a_i) \\ &= (I_n + \alpha \sum_{i=1}^N \tilde{Q}(s_i, a_i) \nabla_\theta g_\theta(s_i, a_i)^\top) \nabla_\phi \theta + \alpha \sum_{i=1}^N g_\theta(s_i, a_i)^\top \nabla_\phi \tilde{Q}(s_i, a_i), \end{aligned} \quad (9)$$

where  $I_n$  is a  $n$ -order identity matrix, and  $n$  is the number of parameters in  $\theta$ . In practice, we can initialize the gradient of the policy parameter with respect to  $\phi$  to a zero matrix and update it according to Equation (9) iteratively. Due to this incremental computing manner, we call the new method *incremental meta-gradient learning* (IMGL). Actually, the above three methods make different tradeoffs between the accuracy of gradient computation and computational complexity. In Appendix, we will provide complexity analysis of these methods and the details of the full algorithm.

## 5 Experiments

We conduct three groups of experiments. The first one is conducted in *cartpole* to verify that our methods can improve the performance of a learning algorithm with beneficial shaping rewards. The



(a) Steps and shaping weights in continuous cartpole      (b) Steps and shaping weights in discrete cartpole

Figure 1: The average step results and the shaping weights of our algorithms in the cartpole tasks

second one is conducted in *MuJoCo* to further test our algorithms in more complex problems. The last one is an adaptability test which shows that our methods can learn good policies and correct shaping weights even when the provided shaping reward function is harmful.

### 5.1 Sparse-Reward Cartpole

In *cartpole*, the agent should apply a force to the pole to keep it from falling. The agent will receive a reward  $-1$  from the environment if the episode ends with the falling of the pole. In other cases, the true reward is zero. The shaping reward for the agent is  $0.1$  if the force applied to the cart and the deviation angle of the pole have the same sign. Otherwise, the shaping reward is zero. In short, such a shaping reward function encourages the actions which make the deviation angle smaller and definitely is beneficial. We adopt the PPO algorithm [16] as the base learner and test our algorithms in both continuous and discrete action settings. We evaluate the three versions of our algorithm, namely BiPaRS-EM, BiPaRS-MGL, and BiPaRS-IMGL and compare them with the naive shaping (NS) method which directly adds the shaping rewards to the original ones, and the dynamic potential-based advice (DPBA) method [6] which transforms an arbitrary reward into a potential value. The details of the *cartpole* problem and the algorithm settings are given in Appendix.

**Test Setting:** The test of each method contains  $1,200,000$  training steps. During the training process, a 20-episode evaluation is conducted every  $4,000$  steps and we record the average steps per episode (ASPE) performance of the tested method at each evaluation point. The ASPE performance stands for how long the agent can keep the pole from falling. The maximal length of an episode is  $200$ , which means that the highest ASPE value that can be achieved is  $200$ . To investigate how our algorithms adjust the shaping weights, we also record the average shaping weights of the experienced states or state-action pairs every  $4,000$  steps. The shaping weights of our methods are initialized to  $1$ .

**Results:** The results of all tests are averaged over  $20$  independent runs using different random seeds and are shown in Figure 1. With the provided shaping reward function, all these methods can improve the learning performance of the PPO algorithm (the left columns in Figs 1(a) and 1(b)). In the continuous-action cartpole task, the performance gap between PPO and the shaping methods is small. In the discrete-action cartpole task, PPO only converges to  $170$ , but with the shaping methods it almost achieves the highest ASPE value  $200$ . By investigating the shaping weight curves in Figure 1, it can be found that our algorithms successfully recognize the positive utility of the given shaping reward function, since all of them keep outputting positive shaping weights during the learning process. For example, in Figure 1(a), the average shaping weights of the BiPaRS-MGL and BiPaRS-IMGL algorithms start from  $1.0$  and finally reach  $1.5$  and  $1.8$ , respectively. The BiPaRS-EM algorithm also learns higher shaping weights than the initial values.

### 5.2 MuJoCo

We choose five MuJoCo tasks *Swimmer-v2*, *Hopper-v2*, *Humanoid-v2*, *Walker2d-v2*, and *HalfCheetah-v2* from OpenAI Gym-v1 to test our algorithms. In each task, the agent is a robot composed of several joints and should decide the amount of torque to apply to each joint in each step. The true reward function is the one predefined in Gym. The shaping reward function is adopted from a recent paper [22] which studies policy optimization with reward constraints and proposes the reward constrained policy optimization (RCPO) algorithm. Specifically, for each state-action pair  $(s, a)$ , we define its shaping reward  $f(s, a) = w(0.25 - \frac{1}{L} \sum_{i=1}^L |a_i|)$ , where  $L$  is the number of joints,  $a_i$  is the torque

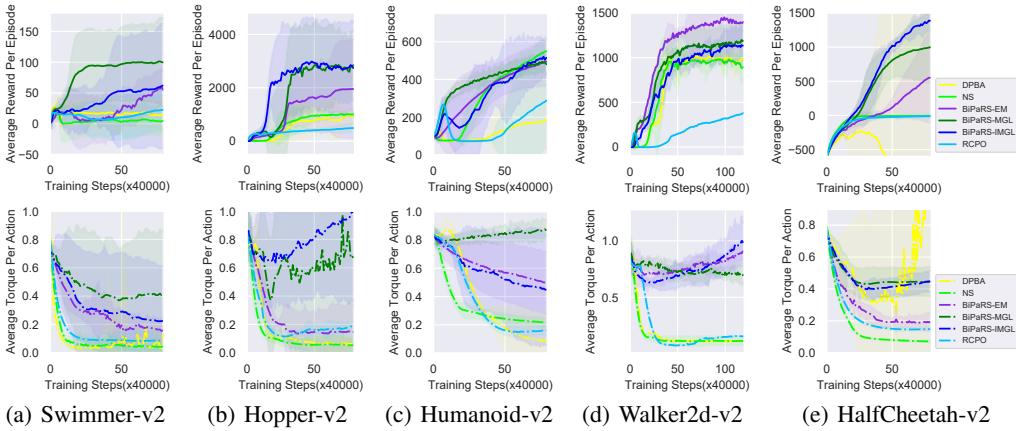


Figure 2: Results of the MuJoCo experiment. The shaded areas are 95% confidence intervals

applied to joint  $i$ , and  $w$  is a task-specific weight (given in Appendix) which makes  $f(s, a)$  have the same scale as the true reward. Concisely,  $f$  requires the average torque amount of each action to be smaller than 0.25, which most probably can result in a sub-optimal policy [22]. Although such shaping reward function is originally used as constraints for achieving safety, we only care about whether our methods can benefit the maximization of the true rewards and how they will do if the shaping rewards are in conflict with the true rewards.

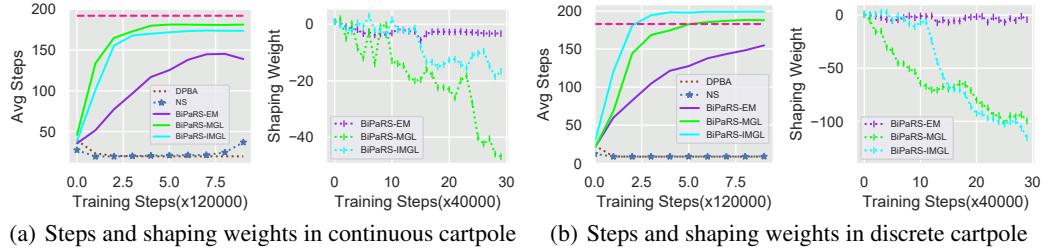
**Test Setting:** We adopt PPO as the base learner and still compare our shaping methods with the NS and DPBA methods. The RCPO algorithm [22] is also adopted as a baseline. The tests in *Walker2d-v2* contain 4,800,000 training steps and the other tests contain 3,200,000. A 20-episode evaluation is conducted every 4,000 training steps. We record the average reward per episode (ARPE) performance and the average torque amount of the agent’s actions at each evaluation point. The maximal length of an episode is 1000.

**Results:** The results of the MuJoCo tests are averaged over 10 runs with different random seeds and are shown in Figure 2. Our algorithms adapt well to the given shaping reward function in each test. For example, in the *Hopper-v2* task (Figure 2(b)), BiPaRS-MGL and BiPaRS-IMGL are far better than the other methods. The BiPaRS-EM method performs slightly worse, but is still better than the baseline methods. The torque amount curves of the tested algorithms are consistent with their reward curves. In some tasks such as *Swimmer-v2* and *HalfCheetah-v2*, the BiPaRS methods slow down the decaying speed of the agent’s torque amount. In the other tasks such as *Hopper-v2* and *Walker2d-v2*, our methods even raise the torque amount values. Interestingly, in Figure 2(b) we can find a v-shaped segment on each of the torque curves of BiPaRS-MGL and BiPaRS-IMGL. This indicates that the two algorithms try to reduce the torque amount of actions at first, but do the opposite after realizing that a small torque amount is unbeneficial. The algorithm performance in our *MuJoCo* test may not match the state-of-the-art in the DRL domain since  $f$  brings additional difficulty for policy optimization. It should also be noted that the goal of this test is to show the disadvantage of fully exploiting the given shaping rewards and our algorithms have the ability to avoid this.

### 5.3 Adaptability Test

Three additional tests are conducted in *cartpole* to further investigate the adaptability of our methods. In the first test, we adopt a harmful shaping reward function which gives the agent a reward  $-0.1$  when the deviation angle of the pole becomes smaller. In the second test, the same shaping reward function is adopted, but our BiPaRS algorithms reuse the shaping weight functions learnt in the first test and only optimize the policy. The goal of the two tests is to show that our methods can identify the harmful shaping rewards and the learnt shaping weights are helpful. In the third test, we use shaping rewards randomly distributed in  $[-1, 1]$  to investigate how our methods adapt to an arbitrary shaping reward function. The other settings are the same as the original *cartpole* experiment.

The results of the first test are shown in Figure 3. The unbeneficial shaping rewards inevitably hurt the policy optimization process, so directly comparing the learning performance of PPO with the



(a) Steps and shaping weights in continuous cartpole      (b) Steps and shaping weights in discrete cartpole

Figure 3: The average step and shaping weight curves of our algorithms in cartpole with a harmful shaping reward function. The magenta lines represent the values that PPO reaches in Figure 1

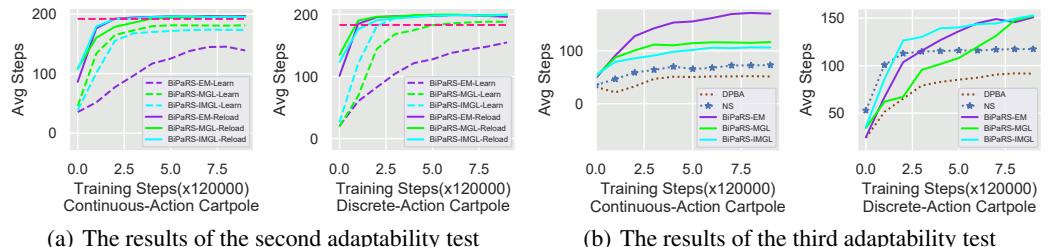


Figure 4: The results of the second adaptability test where the BiPaRS algorithms reload the learnt shaping weight functions and the third adaptability test where the shaping rewards are randomly distributed in  $[-1, 1]$

tested shaping methods is meaningless. Instead, in Figure 3 we plot the step values achieved by PPO in experiment 5.1 as reference (the magenta lines). We can find that the NS and DPBA methods perform poorly since their step curves stay at a low level in the whole training process. In contrast, our methods recognize that the shaping reward function is harmful and have made efforts to reduce its bad influence. For example, in Figure 3(a) BiPaRS-EM finally achieves a decent ASPE value 130, and the step curves of BiPaRS-MGL and BiPaRS-IMGL are very close to the reference line (around 180). In discrete-action *cartpole*, the two methods even achieve higher values than the reference line (Figure 3(b)). The negative shaping weights shown in right columns of Figures 3(a) and 3(b) indicate that our methods are trying to transform the harmful shaping rewards into beneficial ones.

We put the results of the second and third tests in Figures 4(a) and 4(b), respectively. Now examine Figure 4(a), where the dashed lines are the step curves of our methods in the first test. Obviously, the reloaded shaping weight functions enables the BiPaRS methods to learn much better and faster. Furthermore, the learnt policies are better than the policy learnt by PPO, which proves that the learnt shaping weights are correct and helpful. In the third adaptability test, our methods also perform better than the baseline methods. The BiPaRS-EM method achieves the highest step value (about 170) in the continuous-action cartpole, and all our three methods reach the highest step value 150 in the discrete-action cartpole. Note that the third test is no easier than the first one because an algorithm may get stuck with opposite shaping reward signals of similar state-action pairs.

#### 5.4 Learning State-Dependent Shaping Weights

One may have noticed that the shaping reward functions adopted in the above experiments (except the random one in the third adaptability test) are either fully helpful or fully harmful, which means that a uniform shaping weight may also work for adaptive utilization of the shaping rewards. In fact, the shaping weights learnt by our methods differ slightly across the state-action space in the cartpole experiment in Section 5.1 and the first adaptability test in Section 5.3. Learning a state-action-independent shaping weight in the two tests seems sufficient. We implement the baseline method which replaces the shaping weight function  $z_\phi$  with a single shaping weight and test it in the two experiments. It can be found from the corresponding results (Figures 5(a) and 5(b)) that the single-weight method learns better and faster than all the other methods. Although our methods can identify the benefits or harmfulness of a given shaping reward function, it is still unclear whether they have

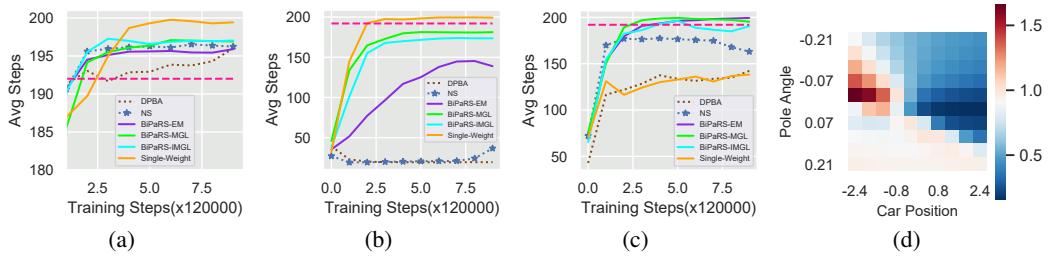


Figure 5: Results of the additional experiments. Figures 5(a) and 5(b) are the results of the single-weight baseline method in continuous-action cartpole with beneficial and harmful shaping rewards, respectively. Figure 5(c) is the result of the experiment with beneficial and harmful shaping rewards in different states, and Figure 5(d) is the heat map of BiPaRS-EM’s shaping weights of 100 states

the ability to learn a state-dependent or state-action-dependent shaping weight function. Therefore, we conduct an additional experiment in the continuous-action cartpole environment with half of the shaping rewards are helpful and the other half of the shaping rewards are harmful. Specifically, the shaping reward is 0.1 if the action taken by the agent reduces the deviation angle when the pole inclines to the right. However, if the action of the agent reduces the deviation angle when the pole inclines to the left, the corresponding shaping reward is  $-0.1$ . The result is shown in Figure 5(c). It can be found that our methods perform the best and the single-weight baseline method cannot perform as well as in Figures 5(a) and 5(b). For better illustration, we also plot the shaping weights learnt by the BiPaRS-EM method across a subset of the state space (containing 100 states) as a heat map in Figure 5(d). Note that a state in the cartpole task is represented by a 4-dimensional vector. For the 100 states used for drawing the heat map, we fix the car velocity (1.0) and pole velocity (0.01) features, and only change the car position and pole angle features. From the heat map figure, we can see that as either of the two state feature changes, the corresponding shaping weight also changes considerably, which indicates that state-dependent shaping weights are learnt.

## 6 Conclusions

In this work, we propose the *bi-level optimization of parameterized reward shaping* (BiPaRS) approach for adaptively utilizing a given shaping reward function. We formulate the utilization problem of shaping rewards, provide formal results for gradient computation, and propose three learning algorithms for solving this problem. The results in cartpole and MuJoCo tasks show that our algorithms can fully exploit beneficial shaping rewards, and meanwhile ignore unbeneficial shaping rewards or even transform them into beneficial ones.

## Broader Impact

Reward design is an important and difficult problem in real-world applications of reinforcement learning (RL). In many cases, researchers or algorithm engineers have some prior knowledge (such as rules and constraints) about the problem to be solved, but cannot represent the knowledge as numeric values very exactly. Improper reward settings may be exploited by an RL algorithm to obtain higher rewards, but with unexpected behaviors learnt. This paper provides an adaptive approach of reward shaping to avoid the repeated and tedious tuning of rewards in RL applications (e.g., video games). A direct impact of our paper is that researchers or algorithm engineers can be liberated from hard work of reward tuning. The shaping weights learnt by our methods indicate the quality of the designed rewards, and thus can help the designers to better understand the problems. Our paper also proposes a general principle for utilizing prior knowledge in the machine learning domain, namely trying to get rid of human cognitive error when the qualitative or rule-based prior knowledge is transformed into numeric values to help with learning.

## Acknowledgments and Disclosure of Funding

We thank the reviewers for the valuable comments and suggestions. Weixun Wang and Jianye Hao are supported by the National Natural Science Foundation of China (Grant Nos.: 61702362, U1836214), Special Program of Artificial Intelligence and Special Program of Artificial Intelligence of Tianjin Municipal Science and Technology Commission (No.: 569 17ZXRGGX00150). Yixiang Wang and Feng Wu are supported in part by the National Natural Science Foundation of China (Grant No. U1613216, Grant No. 61603368).

## References

- [1] Sam Devlin and Daniel Kudenko. Theoretical considerations of potential-based reward shaping for multi-agent systems. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'11)*, pages 225–232, 2011.
- [2] Sam Michael Devlin and Daniel Kudenko. Dynamic potential-based reward shaping. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'12)*, pages 433–440, 2012.
- [3] Marco Dorigo and Marco Colombetti. Robot shaping: Developing autonomous agents through learning. *Artificial intelligence*, 71(2):321–370, 1994.
- [4] Zhao-Yang Fu, De-Chuan Zhan, Xin-Chun Li, and Yi-Xing Lu. Automatic successive reinforcement learning with multiple auxiliary rewards. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI'19)*, pages 2336–2342, 2019.
- [5] Marek Grzes and Daniel Kudenko. Learning potential for reward shaping in reinforcement learning with tile coding. In *Proceedings of the AAMAS 2008 Workshop on Adaptive and Learning Agents and Multi-Agent Systems (ALAMAS-ALAg 2008)*, pages 17–23, 2008.
- [6] Anna Harutyunyan, Sam Devlin, Peter Vrancx, and Ann Nowe. Expressing arbitrary reward functions as potential-based advice. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, pages 2652–2658, 2015.
- [7] Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.
- [8] Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI'17)*, pages 2140–2146, 2017.
- [9] Adam Laud and Gerald DeJong. The influence of reward on the speed of reinforcement learning: An analysis of shaping. In *Proceedings of the 20th International Conference on Machine Learning (ICML'03)*, pages 440–447, 2003.
- [10] Ofir Marom and Benjamin Rosman. Belief reward shaping in reinforcement learning. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI'18)*, pages 3762–3769, 2018.
- [11] Bhaskara Marthi. Automatic shaping and decomposition of reward functions. In *Proceedings of the 24th International Conference on Machine Learning (ICML'07)*, pages 601–608, 2007.
- [12] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the 16th International Conference on Machine Learning (ICML'99)*, pages 278–287, 1999.
- [13] OpenAI. Openai five. <https://blog.openai.com/openai-five/>, 2018.
- [14] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the 34th International Conference on Machine Learning (ICML'17)*, pages 2778–2787, 2017.
- [15] Jette Randløv and Preben Alstrøm. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the 15th International Conference on Machine Learning (ICML'98)*, pages 463–471, 1998.

- [16] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint*, arXiv:1707.06347, 2017.
- [17] Satinder P. Singh, Andrew G. Barto, and Nuttapong Chentanez. Intrinsically motivated reinforcement learning. In *Advances in Neural Information Processing Systems 17 (NIPS'04)*, pages 1281–1288, 2004.
- [18] Shihong Song, Jiayi Weng, Hang Su, Dong Yan, Haosheng Zou, and Jun Zhu. Playing fps games with environment-aware hierarchical reinforcement learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (AAAI'19)*, pages 3475–3482, 2019.
- [19] Jonathan Sorg, Satinder P. Singh, and Richard L. Lewis. Reward design via online gradient ascent. In *Advances in Neural Information Processing Systems 23 (NIPS'10)*, pages 2190–2198, 2010.
- [20] Fan-Yun Sun, Yen-Yu Chang, Yueh-Hua Wu, and Shou-De Lin. Designing non-greedy reinforcement learning agents with diminishing reward shaping. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society (AIES 2018)*, pages 297–302, 2018.
- [21] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of Advances in Neural Information Processing Systems 12 (NIPS'00)*, pages 1057–1063, 1999.
- [22] Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. Reward constrained policy optimization. In *Proceedings of the 7th International Conference on Learning Representations (ICLR'19)*, 2019.
- [23] Eric Wiewiora. Potential-based shaping and q-value initialization are equivalent. *J. Artif. Intell. Res.*, 19:205–208, 2003.
- [24] Eric Wiewiora, Garrison W Cottrell, and Charles Elkan. Principled methods for advising reinforcement learning agents. In *Proceedings of the 20th International Conference on Machine Learning (ICML'03)*, pages 792–799, 2003.
- [25] Yueh-Hua Wu and Shou-De Lin. A low-cost ethics shaping approach for designing reinforcement learning agents. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI'18)*, pages 1687–1694, 2018.
- [26] Zeyu Zheng, Junhyuk Oh, Matteo Hessel, Zhongwen Xu, Manuel Kroiss, Hado van Hasselt, David Silver, and Satinder Singh. What can learned intrinsic rewards capture? In *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*, 2019.
- [27] Zeyu Zheng, Junhyuk Oh, and Satinder Singh. On learning intrinsic rewards for policy gradient methods. In *Advances in Neural Information Processing Systems 31 (NeurIPS'18)*, pages 4649–4659, 2018.
- [28] Haosheng Zou, Tongzheng Ren, Dong Yan, Hang Su, and Jun Zhu. Reward shaping via meta-learning. *arXiv preprint*, arXiv:1901.09330, 2019.

## A Complexity and Algorithm

### A.1 Complexity Analysis

In this section, we provide the complexity analysis of the gradient approximation methods proposed in Section 4 and show the full algorithm for solving the bi-level optimization of parameterized reward shaping (BiPaRS) problem. We denote the number of parameters of the policy function  $\pi_\theta$  by  $n$  and the number of parameters of the shaping weight function  $z_\phi$  by  $m$ , respectively.

**Explicit Mapping:** The explicit mapping (EM) method includes the shaping weight function  $z_\phi$  as an input of  $\pi_\theta$  and approximately computes  $\nabla_\phi \log \pi_\theta(s, a)$  as  $\nabla_z \log \pi_\theta(s, a, z)|_{z=z_\phi(s)} \nabla_\phi z_\phi(s)$ . Therefore, the computational complexity of the EM method is  $\mathcal{O}(m)$ .

**Meta-Gradient Learning:** Given the shaping weight function parameter  $\phi$ , let  $\theta$  and  $\theta'$  denote the policy parameters before and after one round of low-level optimization, and assume that a batch of  $N$  ( $N > 0$ ) samples  $\mathcal{B} = \{(s_i, a_i) | i = 1, \dots, N\}$  is used for updating  $\theta$ . The meta-gradient learning

(MGL) method computes  $\nabla_\phi \theta'$  as

$$\begin{aligned}\nabla_\phi \theta' &\approx \alpha \sum_{i=1}^N g_\theta(s_i, a_i)^\top \nabla_\phi \sum_{t=0}^{|\tau_i|-1} \gamma^t (r_i^t + z_\phi(s_i^t, a_i^t) f(s_i^t, a_i^t)) \\ &= \alpha \sum_{i=1}^N g_\theta(s_i, a_i)^\top \sum_{t=0}^{|\tau_i|-1} \gamma^t f(s_i^t, a_i^t) \nabla_\phi z_\phi(s_i^t, a_i^t).\end{aligned}\tag{A.10}$$

It seems that the computational complexity and space complexity of the (MGL) method are  $\mathcal{O}(Nnm)$ . However, we can reduce both of them to  $\mathcal{O}(N(n+m))$ . Note that Equation (A.10) should be integrated with Theorem 1 for computing the gradient of the expected true reward  $J(z_\phi)$  with respect to  $\phi$  in the upper-level optimization. Without loss of generality, we assume only one state-action pair  $(s, a)$  is used to update  $\phi$  under the new policy  $\theta'$ . By substituting Equation (A.10) into Theorem 1, we have

$$\begin{aligned}\nabla_\phi J(z_\phi) &\approx \nabla_\phi \log \pi_{\theta'}(s, a) Q(s, a) \\ &= \nabla_{\theta'} \log \pi_{\theta'}(s, a) \nabla_\phi \theta' Q(s, a) \\ &\approx \alpha \nabla_{\theta'} \log \pi_{\theta'}(s, a) \left( \sum_{i=1}^N g_\theta(s_i, a_i)^\top \sum_{t=0}^{|\tau_i|-1} \gamma^t f(s_i^t, a_i^t) \nabla_\phi z_\phi(s_i^t, a_i^t) \right) Q(s, a),\end{aligned}\tag{A.11}$$

where  $Q(s, a)$  is the state-action value function under the policy  $\pi_{\theta'}$ . Actually, we can change the computing order of Equation (A.11) and compute it as

$$\nabla_\phi J(z_\phi) \approx \alpha \sum_{i=1}^N (\nabla_{\theta'} \log \pi_{\theta'}(s, a) Q(s, a) g_\theta(s_i, a_i)^\top) \sum_{t=0}^{|\tau_i|-1} \gamma^t f(s_i^t, a_i^t) \nabla_\phi z_\phi(s_i^t, a_i^t).\tag{A.12}$$

For each  $i$  in the sum loop of this equation, computing the term  $\nabla_{\theta'} \log \pi_{\theta'}(s, a) Q(s, a) g_\theta(s_i, a_i)^\top$  costs  $\mathcal{O}(n)$  time and space, computing the term  $\sum_{t=0}^{|\tau_i|-1} \gamma^t f(s_i^t, a_i^t) \nabla_\phi z_\phi(s_i^t, a_i^t)$  costs  $\mathcal{O}(m)$  time and space (we treat  $|\tau_i|$  as a constant), and computing the product of these two terms also costs  $\mathcal{O}(m)$  time and space. Therefore, the space complexity and computational complexity for computing Equation (A.12) are  $\mathcal{O}(N(n+m))$ .

**Incremental Meta-Gradient Learning:** The incremental meta-gradient learning (IMGL) is a generalized version of the MGL method. We still let  $\theta$  and  $\theta'$  denote the policy parameters before and after one round of low-level optimization, and assume that a batch of  $N$  ( $N > 0$ ) samples  $\mathcal{B} = \{(s_i, a_i) | i = 1, \dots, N\}$  is used to update  $\theta$ . The IMGL method computes  $\nabla_\phi \theta'$  as

$$\begin{aligned}\nabla_\phi \theta' &= \nabla_\phi \theta + \alpha \sum_{i=1}^N \nabla_\phi g_\theta(s_i, a_i)^\top \tilde{Q}(s_i, a_i) + \alpha \sum_{i=1}^N g_\theta(s_i, a_i)^\top \nabla_\phi \tilde{Q}(s_i, a_i) \\ &= (I_n + \alpha \sum_{i=1}^N \tilde{Q}(s_i, a_i) \nabla_\theta g_\theta(s_i, a_i)^\top) \nabla_\phi \theta + \alpha \sum_{i=1}^N g_\theta(s_i, a_i)^\top \nabla_\phi \tilde{Q}(s_i, a_i),\end{aligned}\tag{A.13}$$

where  $I_n$  is a  $n$ -order identity matrix and  $\tilde{Q}$  denotes the state-action value function in the modified MDP. Compared with the EM and MGL methods, IMGL is much more computationally expensive since the term  $\alpha \sum_{i=1}^N \tilde{Q}(s_i, a_i) \nabla_\theta g_\theta(s_i, a_i)^\top$  involves the computation of the Hessian matrix  $\nabla_\theta g_\theta(s_i, a_i)^\top$  for each  $i$ , which has  $\mathcal{O}(Nn^3)$  computational complexity. However, there are several ways for reducing the high computational cost of IMGL. Firstly, the Hessian matrix  $\nabla_\theta g_\theta(s_i, a_i)^\top$  can be approximated using outer product of gradients (OPG) estimate. Secondly, for simple problems, we can use a small policy model with a few parameters. Lastly, we can even omit the second-order term  $\alpha \sum_{i=1}^N \tilde{Q}(s_i, a_i) \nabla_\theta g_\theta(s_i, a_i)^\top$  to get another approximation of  $\nabla_\phi \theta'$ .

## A.2 The Full Algorithm

The main workflow of the BiPaRS framework can be illustrated by Figure 3. In this figure, the left column shows the training loop of the agent's policy, where the expert knowledge (i.e., shaping reward) is integrated with the samples generated from the agent-environment interaction. The central

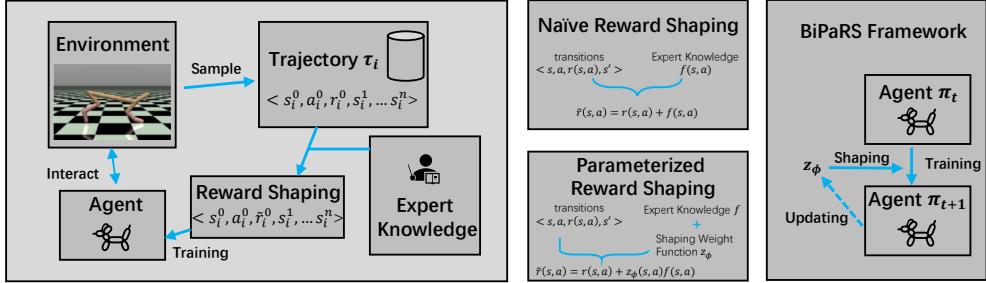


Figure 3: An overview of the BiPaRS framework

column shows the difference between the parameterized reward shaping and the traditional reward shaping methods, namely the shaping weight function  $z_\phi$  for adaptive utilization of the shaping reward function  $f$ . The right column of the figure intuitively shows the alternating optimization process of the policy and shaping weight function.

Now we summarize all the methods proposed in our paper into the general learning algorithm in Algorithm 1. This algorithm actually corresponds to three specific learning algorithms which adopt the explicit mapping (EM), meta-gradient learning (MGL), and incremental meta-gradient learning (IMGL) methods for gradient approximation, respectively. As shown in the algorithm table, the policy parameter  $\theta$  and the parameter of the shaping weight function  $\phi$  are optimized iteratively. At each iteration  $t$ ,  $\theta$  is firstly updated according to the shaping rewards (lines 5 to 14), which are weighted by the shaping weight function  $z_\phi$ . If MGL or IMGL is chosen as the method for gradient approximation, then the meta-gradient  $\nabla_\phi \theta$ , which is denoted by the variable  $h$ , will be computed at the same time (lines 9 to 14), where  $\nabla_\theta \log \pi_\theta(s, a)$  is simplified as  $g_\theta(s, a)$ . After updating  $\theta$ ,  $\phi$  is updated based on the true rewards sampled by the new policy  $\pi_\theta$  and the approximated gradient of  $\pi_\theta$  with respect to  $\phi$  (lines 15 to 22). For the EM method (line 21), we do not explicitly represent  $\pi_\theta$  as a function of  $z_\phi$  in order to keep the consistency of notations. We also omit some details in Algorithm 1, such as the learning of the two value functions  $Q$  and  $\tilde{Q}$ , and the computation of the gradient  $\nabla_\phi \tilde{Q}(s, a)$ .

## B Theorem Proof

This section gives the proof of Theorem 1. We make the following assumptions.

**Assumption 1.** Let  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, r, p_0, \gamma \rangle$  denote the original MDP,  $\pi_\theta$  be the policy, and  $z_\phi$  be the shaping weight function in the BiPaRS problem, respectively. We assume that  $P(s, a, s')$ ,  $r(s, a)$ ,  $p_0(s)$  are continuous w.r.t. the variables  $s$ ,  $a$ , and  $s'$ , and  $\pi_\theta(s, a)$  are continuous w.r.t.  $s$ ,  $a$ ,  $\theta$ , and  $\phi$ .

**Assumption 2.** For the MDP  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, r, p_0, \gamma \rangle$ , there exists a real number  $b$  such that  $r(s, a) < b, \forall (s, a)$ .

**Theorem 2.** Given the shaping weight function  $z_\phi$  and the stochastic policy  $\pi_\theta$  of the agent in the upper level of the BiPaRS problem (Equation (2) in the paper), the gradient of the objective function  $J(z_\phi)$  with respect to the variable  $\phi$  is

$$\nabla_\phi J(z_\phi) = \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta} [\nabla_\phi \log \pi_\theta(s, a) Q^\pi(s, a)], \quad (\text{B.14})$$

where  $Q^\pi$  is the state-action value function of  $\pi_\theta$  in the original MDP.

*Proof Sketch.* Our proof is similar to that of the stochastic policy gradient theorem (Sutton et al. (1999)) and the deterministic policy gradient theorem (Silver et al. (2014)). Given a stochastic policy  $\pi_\theta$ , let  $V^\pi$  and  $Q^\pi$  denote the state value function and state-action value function of  $\pi$  in the original MDP  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, r, p_0, \gamma \rangle$ , respectively. Note that Assumption 1 implies that  $V^\pi$  and  $Q^\pi$  are continuous functions of  $\phi$  and Assumption 2 guarantees that  $V^\pi(s)$ ,  $Q^\pi(s, a)$ ,  $\nabla_\phi V^\pi(s)$  and  $\nabla_\phi Q^\pi(s, a)$  are bounded for any  $s \in \mathcal{S}$  and any  $a \in \mathcal{A}$ . In our proof, the two assumptions are necessary to exchange integrals and derivatives, and the integration orders.

---

**Algorithm 1:** Bilevel Optimization of Parameterized Reward Shaping (BiPaRS)

---

**Input:** Learning rates  $\alpha_\theta$  and  $\alpha_\phi$

- 1 Initialize the policy parameter  $\theta$  and the shaping weight function parameter  $\phi$ ;
- 2 Initialize the true and shaping value functions  $Q$  and  $\tilde{Q}$ ;
- 3 Initialize the meta-gradient  $h$  to a zero matrix;
- 4 **for**  $t = 1, 2, \dots$ , **do**
- 5     Run policy  $\pi_\theta$  in the modified MDP with  $z_\phi$ ;
- 6      $\mathcal{T}' \leftarrow$  the set of sampled experiences;
- 7     Update  $\tilde{Q}$  using samples from  $\mathcal{T}'$ ;
- 8      $\theta' \leftarrow \theta + \alpha_\theta \sum_{(s,a) \sim \mathcal{T}'} \nabla_\theta \log \pi_\theta(s, a) \tilde{Q}(s, a)$ ;
- 9     **if**  $MGL$  is used **then**
- 10          $h \leftarrow \alpha_\theta \sum_{(s,a) \sim \mathcal{T}'} g_\theta(s, a)^\top \nabla_\phi \tilde{Q}(s, a)$ ;
- 11     **else if**  $IMGL$  is used **then**
- 12          $h_1 \leftarrow (\sum_{(s,a) \sim \mathcal{T}'} \nabla_\theta g_\theta(s, a)^\top \tilde{Q}(s, a)) h$ ;
- 13          $h_2 \leftarrow \sum_{(s,a) \sim \mathcal{T}'} g_\theta(s, a) \nabla_\phi \tilde{Q}(s, a)$ ;
- 14          $h \leftarrow h + \alpha_\theta (h_1 + h_2)$ ;
- 15     Run policy  $\pi_{\theta'}$  in the original MDP;
- 16      $\mathcal{T} \leftarrow$  the set of sampled experiences;
- 17     Update  $Q$  using samples from  $\mathcal{T}$ ;
- 18     **if**  $MGL$  or  $IMGL$  is used **then**
- 19          $\Delta\phi \leftarrow \sum_{(s,a) \sim \mathcal{T}} \nabla_{\theta'} \log \pi_{\theta'}(s, a) Q(s, a) h$ ;
- 20     **else**
- 21          $\Delta\phi \leftarrow \sum_{(s,a) \sim \mathcal{T}} \nabla_z \log \pi_{\theta'}(s, a) \nabla_\phi z_\phi(s) Q(s, a)$ ;
- 22      $\phi \leftarrow \phi + \alpha_\phi \Delta\phi$ ;
- 23      $\theta \leftarrow \theta'$ ;

---

Obviously, for any state  $s$  we have

$$V^\pi(s) = \int_{\mathcal{A}} \pi_\theta(s, a) Q^\pi(s, a) da.$$

Therefore, the gradient of  $V^\pi(s)$  with respect to the shaping weight function parameter  $\phi$  is

$$\begin{aligned} \nabla_\phi V^\pi(s) &= \nabla_\phi \left( \int_{\mathcal{A}} \pi_\theta(s, a) Q^\pi(s, a) da \right) \\ &= \int_{\mathcal{A}} \left( \nabla_\phi \pi_\theta(s, a) Q^\pi(s, a) + \pi_\theta(s, a) \nabla_\phi Q^\pi(s, a) \right) da. \end{aligned}$$

The above equation is an application of the Leibniz integral rule to exchange the orders of derivative and integral, which requires Assumption 1.

By further expanding the term  $\nabla_\phi Q^\pi(s, a)$ , we can obtain

$$\begin{aligned} \nabla_\phi V^\pi(s) &= \int_{\mathcal{A}} \left( \nabla_\phi \pi_\theta(s, a) Q^\pi(s, a) + \pi_\theta(s, a) \nabla_\phi Q^\pi(s, a) \right) da \\ &= \int_{\mathcal{A}} \left( \nabla_\phi \pi_\theta(s, a) Q^\pi(s, a) + \pi_\theta(s, a) \nabla_\phi (r(s, a) + \gamma \int_{\mathcal{S}} P(s, a, s') V^\pi(s') ds') \right) da \\ &= \int_{\mathcal{A}} \nabla_\phi \pi_\theta(s, a) Q^\pi(s, a) da + \int_{\mathcal{A}} \pi_\theta(s, a) \int_{\mathcal{S}} \gamma P(s, a, s') \nabla_\phi V^\pi(s') ds' da \\ &= \int_{\mathcal{A}} \nabla_\phi \pi_\theta(s, a) Q^\pi(s, a) da + \int_{\mathcal{S}} \int_{\mathcal{A}} \gamma \pi_\theta(s, a) P(s, a, s') da \nabla_\phi V^\pi(s') ds' \\ &= \int_{\mathcal{A}} \nabla_\phi \pi_\theta(s, a) Q^\pi(s, a) da + \int_{\mathcal{S}} \gamma p(s \rightarrow s', 1, \pi_\theta) \nabla_\phi V^\pi(s') ds', \end{aligned}$$

where  $p(s' \rightarrow s, t, \pi_\theta)$  is the probability that state  $s$  is visited after  $t$  steps from state  $s'$  under the policy  $\pi_\theta$ . In the above derivation, the first step is an expansion of the Bellman equation. The second

step is by exchanging the order of derivative and integral. The third step exchanges the order of integration by using Fubini's Theorem, which requires our assumptions. The last step is according to the definition of  $p$ . By expanding  $\nabla_\phi V^\pi(s')$  in the same way, we can get

$$\begin{aligned}
\nabla_\phi V^\pi(s) &= \int_{\mathcal{A}} \nabla_\phi \pi_\theta(s, a) Q^\pi(s, a) da + \int_{\mathcal{S}} \gamma p(s \rightarrow s', 1, \pi_\theta) \nabla_\phi \left( \int_{\mathcal{A}} \pi_\theta(s', a') Q^\pi(s', a') da' \right) ds' \\
&= \int_{\mathcal{A}} \nabla_\phi \pi_\theta(s, a) Q^\pi(s, a) da + \int_{\mathcal{S}} \gamma p(s \rightarrow s', 1, \pi_\theta) \int_{\mathcal{A}} \nabla_\phi \pi_\theta(s', a') Q^\pi(s', a') da' ds' \\
&\quad + \int_{\mathcal{S}} \gamma p(s \rightarrow s', 1, \pi_\theta) \int_{\mathcal{A}} \pi_\theta(s', a') \nabla_\phi Q^\pi(s', a') da' ds' \\
&= \int_{\mathcal{A}} \nabla_\phi \pi_\theta(s, a) Q^\pi(s, a) da + \int_{\mathcal{S}} \gamma p(s \rightarrow s', 1, \pi_\theta) \int_{\mathcal{A}} \nabla_\phi \pi_\theta(s', a') Q^\pi(s', a') da' ds' \\
&\quad + \int_{\mathcal{S}} \gamma p(s \rightarrow s', 1, \pi_\theta) \int_{\mathcal{S}} \gamma p(s' \rightarrow s'', 1, \pi_\theta) \nabla_\phi V^\pi(s'') ds'' ds' \\
&= \int_{\mathcal{A}} \nabla_\phi \pi_\theta(s, a) Q^\pi(s, a) da + \int_{\mathcal{S}} \gamma p(s \rightarrow s', 1, \pi_\theta) \int_{\mathcal{A}} \nabla_\phi \pi_\theta(s', a') Q^\pi(s', a') da' ds' \\
&\quad + \int_{\mathcal{S}} \gamma p(s \rightarrow s', 2, \pi_\theta) \nabla_\phi V^\pi(s') ds' \\
&= \dots \\
&= \sum_{t=0}^{\infty} \int_{\mathcal{S}} \gamma^t p(s \rightarrow s', t, \pi_\theta) \int_{\mathcal{A}} \nabla_\phi \pi_\theta(s', a) Q^\pi(s', a) da ds' \\
&= \int_{\mathcal{S}} \sum_{t=0}^{\infty} \gamma^t p(s \rightarrow s', t, \pi_\theta) \int_{\mathcal{A}} \nabla_\phi \pi_\theta(s', a) Q^\pi(s', a) da ds'.
\end{aligned}$$

Recall that the upper-level objective  $J(z_\phi) = \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} r(s, a) da ds = \int_{\mathcal{S}} p_0(s) V^\pi(s) ds$ . Therefore, we have

$$\begin{aligned}
\nabla_\phi J(z_\phi) &= \int_{\mathcal{S}} p_0(s) \nabla_\phi V^\pi(s) ds \\
&= \int_{\mathcal{S}} p_0(s) \int_{\mathcal{S}} \sum_{t=0}^{\infty} \gamma^t p(s \rightarrow s', t, \pi_\theta) \int_{\mathcal{A}} \nabla_\phi \pi_\theta(s', a) Q^\pi(s', a) da ds' ds \\
&= \int_{\mathcal{S}} p_0(s) \int_{\mathcal{S}} \sum_{t=0}^{\infty} \gamma^t p(s \rightarrow s', t, \pi_\theta) ds \int_{\mathcal{A}} \nabla_\phi \pi_\theta(s', a) Q^\pi(s', a) da ds' \\
&= \int_{\mathcal{S}} \rho^\pi(s') \int_{\mathcal{A}} \nabla_\phi \pi_\theta(s', a) Q^\pi(s', a) da ds'.
\end{aligned}$$

By using the log-derivative trick, we can finally obtain Equation (B.14).  $\square$

## C BiPaRS for Deterministic Policy Setting

In this section, we define the BiPaRS problem for deterministic policy gradient algorithms. We provide a theorem similar to Theorem 2 and give the proof. Then we show the three methods explicit mapping (EM), meta-gradient learning (MGL), and incremental meta-gradient learning (IMGL) for approximating the gradient of the expected true reward with respect to the shaping weight function parameter  $\phi$  in the deterministic policy setting.

Let  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, r, p_0, \gamma \rangle$  denote an MDP,  $\mu_\theta$  denote an agent's deterministic policy with parameter  $\theta$ ,  $f$  denote the shaping reward function, and  $z_\phi$  denote the shaping weight function with parameter  $\phi$ . Given  $f$  and  $z_\phi$ , the agent should maximize the expected modified reward  $\tilde{J}(\mu_\theta) = \mathbb{E}_{s \sim \rho^\mu} [(r(s, a) + z_\phi(s, a)f(s, a))|_{a=\mu_\theta(s)}]$ . The objective of the shaping weight function  $z_\phi$  is the expected accumulative true reward  $J(z_\phi) = \mathbb{E}_{s \sim \rho^\mu} [r(s, a)|_{a=\mu_\theta(s)}]$ . Formally, the bi-level optimization of parameterized reward shaping (BiPaRS) problem for the deterministic policy setting can be defined as

$$\begin{aligned} & \max_{\phi} \mathbb{E}_{s \sim \rho^\mu} [r(s, a)|_{a=\mu_\theta(s)}] \\ & \text{s.t. } \phi \in \Phi \\ & \theta = \arg \max_{\theta'} \mathbb{E}_{s \sim \rho^\mu} [(r(s, a) + z_\phi(s, a)f(s, a))|_{a=\mu_{\theta'}(s)}] \\ & \text{s.t. } \theta' \in \Theta. \end{aligned} \tag{C.15}$$

The following theorem shows how to compute the gradient of the upper-level objective  $J(z_\phi)$  with respect to the variable  $\phi$  in Equation (C.15).

**Theorem 3.** *Given the shaping weight function  $z_\phi$  and the deterministic policy  $\mu_\theta$  of the agent in the upper level of the BiPaRS problem Equation (C.15), the gradient of the objective function  $J(z_\phi)$  with respect to the shaping weight function parameter  $\phi$  is*

$$\nabla_\phi J(z_\phi) = \mathbb{E}_{s \sim \rho^\mu} [\nabla_\phi \mu_\theta(s) \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)}], \tag{C.16}$$

where  $Q^\mu$  is the state-action value function of  $\mu_\theta$  in the original MDP.

To prove the Theorem, we assume that  $P(s, a, s')$ ,  $\nabla_a P(s, a, s')$ ,  $r(s, a)$ ,  $\nabla_a r(s, a)$ ,  $p_0(s)$  are continuous w.r.t. the variables  $s$ ,  $a$ , and  $s'$ , and  $\mu_\theta(s)$  are continuous w.r.t.  $s$ ,  $\theta$ , and  $\phi$ . We also assume that the rewards are bounded. The proof is as follows.

*Proof Sketch.* Given a deterministic policy  $\mu_\theta$ , for any state  $s$ , the gradient of the state value  $V^\mu(s)$  with respect to  $\phi$  is

$$\begin{aligned} \nabla_\phi V^\mu(s) &= \nabla_\phi Q^\mu(s, \mu_\theta(s)) \\ &= \nabla_\phi \left( r(s, \mu_\theta(s)) + \gamma \int_{\mathcal{S}} P(s, \mu_\theta(s), s') V^\mu(s') ds' \right) \\ &= \nabla_\phi \mu_\theta(s) \nabla_a r(s, a)|_{a=\mu_\theta(s)} + \nabla_\phi \left( \gamma \int_{\mathcal{S}} P(s, \mu_\theta(s), s') V^\mu(s') ds' \right) \\ &= \nabla_\phi \mu_\theta(s) \nabla_a r(s, a)|_{a=\mu_\theta(s)} + \gamma \int_{\mathcal{S}} \nabla_\phi \mu_\theta(s) \nabla_a P(s, a, s')|_{a=\mu_\theta(s)} V^\mu(s') ds' \\ &\quad + \gamma \int_{\mathcal{S}} P(s, \mu_\theta(s), s') \nabla_\phi V^\mu(s') ds' \\ &= \nabla_\phi \mu_\theta(s) \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)} + \int_{\mathcal{S}} \gamma p(s \rightarrow s', 1, \mu_\theta) \nabla_\phi V^\mu(s') ds'. \end{aligned}$$

By expanding  $\nabla_\phi V^\mu(s')$ , we have

$$\begin{aligned}
\nabla_\phi V^\mu(s) &= \nabla_\phi \mu_\theta(s) \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)} + \int_{\mathcal{S}} \gamma p(s \rightarrow s', 1, \mu_\theta) \nabla_\phi Q^\mu(s', \mu_\theta(s')) ds' \\
&= \nabla_\phi \mu_\theta(s) \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)} + \int_{\mathcal{S}} \gamma p(s \rightarrow s', 1, \mu_\theta) \nabla_\phi \mu_\theta(s') \nabla_a Q^\mu(s', a)|_{a=\mu_\theta(s')} ds' \\
&\quad + \int_{\mathcal{S}} \gamma p(s \rightarrow s', 1, \mu_\theta) \int_{\mathcal{S}} \gamma p(s' \rightarrow s'', 1, \mu_\theta) \nabla_\phi V^\mu(s'') ds'' ds' \\
&= \nabla_\phi \mu_\theta(s) \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)} \\
&\quad + \int_{\mathcal{S}} \gamma p(s \rightarrow s', 1, \mu_\theta) \nabla_\phi \mu_\theta(s') \nabla_a Q^\mu(s', a)|_{a=\mu_\theta(s')} ds' \\
&\quad + \int_{\mathcal{S}} \gamma^2 p(s \rightarrow s', 2, \mu_\theta) \nabla_\phi \mu_\theta(s') \nabla_a Q^\mu(s', a)|_{a=\mu_\theta(s')} ds' \\
&\quad + \dots \\
&= \int_{\mathcal{S}} \sum_{t=0}^{\infty} \gamma^t p(s \rightarrow s', t, \mu_\theta) \nabla_\phi \mu_\theta(s') \nabla_a Q^\mu(s', a)|_{a=\mu_\theta(s')} ds'.
\end{aligned}$$

Taking expectation over the initial states, we can obtain

$$\begin{aligned}
\nabla_\phi J(z_\phi) &= \int_{\mathcal{S}} p_0(s) \nabla_\phi V^\mu(s) ds \\
&= \int_{\mathcal{S}} p_0(s) \int_{\mathcal{S}} \sum_{t=0}^{\infty} \gamma^t p(s \rightarrow s', t, \mu_\theta) \nabla_\phi \mu_\theta(s') \nabla_a Q^\mu(s', a)|_{a=\mu_\theta(s')} ds' ds \\
&= \int_{\mathcal{S}} p_0(s) \int_{\mathcal{S}} \sum_{t=0}^{\infty} \gamma^t p(s \rightarrow s', t, \mu_\theta) ds \nabla_\phi \mu_\theta(s') \nabla_a Q^\mu(s', a)|_{a=\mu_\theta(s')} ds' \\
&= \int_{\mathcal{S}} \rho^\mu(s) \nabla_\phi \mu_\theta(s) \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)} ds,
\end{aligned}$$

which is exactly Equation (C.16).  $\square$

### C.1 Explicit Mapping

The explicit mapping method makes the shaping weight function  $z_\phi$  an input of the policy  $\mu_\theta$ . Specifically, the policy  $\mu_\theta$  is redefined as a hyper policy  $\mu_\theta : \mathcal{S}_z \rightarrow \mathcal{A}$ , where  $\mathcal{S}_z = \{(s, z_\phi(s)) | \forall s \in \mathcal{S}\}$ . According to the chain rule, we have

$$\begin{aligned}
\nabla_\phi J(z_\phi) &= \mathbb{E}_{s \sim \rho^\mu} [\nabla_\phi \mu_\theta(s, z) \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s, z), z=z_\phi(s)}] \\
&= \mathbb{E}_{s \sim \rho^\mu} [\nabla_z \mu_\theta(s, z) \nabla_\phi z_\phi(s) \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s, z), z=z_\phi(s)}].
\end{aligned} \tag{C.17}$$

### C.2 Meta-Gradient Learning

Let  $\theta$  and  $\theta'$  be the policy parameters before and after one round of low-level optimization, respectively. Let  $\tilde{Q}$  denote the state-action value function under the policy  $\mu_\theta$  in the modified MDP  $\mathcal{M}' = \langle \mathcal{S}, \mathcal{A}, P, \tilde{r}, p_0, \gamma \rangle$ . We still assume that a batch of  $N$  ( $N > 0$ ) samples  $\mathcal{B} = \{(s_i, a_i) | i = 1, \dots, N\}$  is used to update  $\theta$ . According to the deterministic policy gradient theorem, we have

$$\theta' = \theta + \alpha \sum_{i=1}^N \nabla_\theta \mu_\theta(s_i) \nabla_a \tilde{Q}(s_i, a)|_{a=\mu_\theta(s_i)}, \tag{C.18}$$

where  $\alpha$  is the learning rate. By taking the gradient of the both sides of Equation (C.18) with respect to  $\phi$ , we get

$$\begin{aligned}\nabla_\phi \theta' &= \nabla_\phi \left( \theta + \alpha \sum_{i=1}^N \nabla_\theta \mu_\theta(s_i) \nabla_a \tilde{Q}(s_i, a)|_{a=\mu_\theta(s_i)} \right) \\ &\approx \alpha \sum_{i=1}^N \nabla_\theta \mu_\theta(s_i)^\top \nabla_\phi (\nabla_a \tilde{Q}(s_i, a)|_{a=\mu_\theta(s_i)}),\end{aligned}\tag{C.19}$$

where  $\theta$  is treated as a constant with respect to  $\phi$ . However, for each sample  $i$  in the batch  $\mathcal{B}$ , we cannot directly compute the value of the term  $\nabla_\phi (\nabla_a \tilde{Q}(s_i, a)|_{a=\mu_\theta(s_i)})$  even we replace  $\tilde{Q}(s_i, a)$  by a Monte Carlo return as in the stochastic policy case. We adopt the idea of the explicit mapping method to solve this problem. That is, to include  $z_\phi(s)$  as an input of  $\tilde{Q}$ . As we discuss in Section 4.1, this makes  $\tilde{Q}$  the state-action value function of  $\mu_\theta$  in an equivalent MDP  $\tilde{\mathcal{M}}_z = \langle \mathcal{S}_z, \mathcal{A}, P_z, \tilde{r}_z, p_z, \gamma \rangle$  and is transparent to the agent. For simplicity, we denote  $\delta(s_i, a, z) = \nabla_a \tilde{Q}(s_i, a, z)$ . With the extended state-action value function, we have

$$\begin{aligned}\nabla_\phi \theta' &\approx \alpha \sum_{i=1}^N \nabla_\theta \mu_\theta(s_i)^\top \nabla_\phi (\nabla_a \tilde{Q}(s_i, a, z)|_{a=\mu_\theta(s_i), z=z_\phi(s_i)}) \\ &= \alpha \sum_{i=1}^N \nabla_\theta \mu_\theta(s_i)^\top \nabla_\phi \delta(s_i, a, z)|_{a=\mu_\theta(s_i), z=z_\phi(s_i)} \\ &= \alpha \sum_{i=1}^N \nabla_\theta \mu_\theta(s_i)^\top \nabla_z \delta(s_i, a, z)|_{a=\mu_\theta(s_i), z=z_\phi(s_i)} \nabla_\phi z_\phi(s_i).\end{aligned}\tag{C.20}$$

### C.3 Incremental Meta-Gradient Learning

In Equation (C.19), we can also treat  $\theta$  as a non-constant with respect to  $\phi$  because  $\phi$  is optimized according to the old policy  $\pi_\theta$  in the last round of upper-level optimization. For the simplification purpose, we let  $g_\theta(s) = \nabla_\theta \mu_\theta(s)$ . Then Equation (C.19) can be rewritten as

$$\begin{aligned}\nabla_\phi \theta' &= \nabla_\phi \left( \theta + \alpha \sum_{i=1}^N \nabla_\theta \mu_\theta(s_i) \nabla_a \tilde{Q}(s_i, a)|_{a=\mu_\theta(s_i)} \right) \\ &= \nabla_\phi \theta + \alpha \sum_{i=1}^N \left( \nabla_\phi g_\theta(s_i)^\top \nabla_a \tilde{Q}(s_i, a) + g_\theta(s_i)^\top \nabla_\phi (\nabla_a \tilde{Q}(s_i, a)) \right) \Big|_{a=\mu_\theta(s_i)}\end{aligned}\tag{C.21}$$

For computing the value of  $\nabla_\phi (\nabla_a \tilde{Q}(s_i, a)|_{a=\mu_\theta(s_i)})$ , once again we can include  $z_\phi$  in the input of  $\tilde{Q}$ . Denoting  $\nabla_a \tilde{Q}(s_i, a, z)$  by  $\delta(s_i, a, z)$ , we have

$$\nabla_\phi \theta' \approx \nabla_\phi \theta + \alpha \sum_{i=1}^N \left( \nabla_\theta g_\theta(s_i)^\top \nabla_\phi \theta \delta(s_i, a, z) + g_\theta(s_i)^\top \nabla_\phi \delta(s_i, a, z) \right) \Big|_{a=\mu_\theta(s_i), z=z_\phi(s_i)}\tag{C.22}$$

By substituting Equation (C.22) into Equation (C.16), we can get the third approximation of the gradient  $\nabla_\phi J(z_\phi)$ . In fact, in Equation (C.21) we can treat the shaping state-action value  $\tilde{Q}$  as a constant with respect to  $\phi$  so that we do not have to extend the input space of  $\tilde{Q}$  and can get another version of  $\nabla_\phi \theta'$ , namely

$$\nabla_\phi \theta' \approx \nabla_\phi \theta + \alpha \sum_{i=1}^N \nabla_\theta g_\theta(s_i)^\top \nabla_\phi \theta \nabla_a \tilde{Q}(s_i, a)|_{a=\mu_\theta(s_i)}.\tag{C.23}$$

Furthermore, we can also remove the second-order term  $\alpha \sum_{i=1}^N \nabla_\theta g_\theta(s_i)^\top \nabla_\phi \theta \delta(s_i, a, z)$  of Equation (C.22) and get a computationally cheaper approximation

$$\nabla_\phi \theta' \approx \nabla_\phi \theta + \alpha \sum_{i=1}^N \left( g_\theta(s_i)^\top \nabla_z \delta(s_i, a, z) \nabla_\phi z_\phi(s_i) \right) \Big|_{a=\mu_\theta(s_i), z=z_\phi(s_i)}.\tag{C.24}$$

## D Experiments

In this section, we provide the details of the problem and algorithm hyperparameter settings of the *cartpole* and *MuJoCo* experiments.

### D.1 Cartpole

**Problem Setting:** We choose the *cartpole* task from the OpenAI Gym-v1 benchmark. The cartpole system consists of a pole and a cart. The pole is connected to the cart by an un-actuated joint and the cart can be controlled by the agent to move along the horizontal axis. Each episode starts by setting the position of the cart randomly within the interval  $[-0.05, 0.05]$  and setting the angle between the pole and the vertical direction smaller than 3 degrees. In each step of an episode, the agent should apply a positive or negative force to the cart to let the pole remain within 12 degrees from the vertical direction and keep the position of the cart within  $[-2.4, 2.4]$ . An episode will be terminated if either of the two conditions is broken or the episode has lasted for 200 steps. In the discrete-action cartpole, the action space of the agent has only two actions +1 and -1, while in the continuous-action cartpole, the agent has to decide the specific value of the force to be applied.

**Hyperparameter Settings:** In the *cartpole* experiment, the base learner PPO adopts a two-layer policy network with 8 units in each layer and a two-layer value function network with 32 units in each hidden layer. Both the policy and value function networks adopt *relu* as the activation function in each hidden layer. The policy and value function networks are updated every 20,000 steps and one such update contains 50 optimizing epochs with batch size 1024. The threshold for clipping the probability ratio is 0.5. We adopt the generalized advantage estimator (GAE) to replace Q-function for computing policy gradient and the hyperparameter  $\lambda$  for bias-variance trade-off is 0.95.

The DPBA method adopts a neural network to learn potentials from shaping rewards, which has two full-connected (FC) *tanh* hidden layers with 16 units in the first layer and 8 units in the second layer. The BiPaRS methods also use two-layer neural network to represent the shaping weight function, with 16 and 8 units in the first and second layers and *tanh* as the activation in both layers. The outputs of the shaping weight function network for all state-action pairs are initialized to 1 using the following way. Firstly, the weights and biases of the hidden layers are initialized from a uniform distribution in  $[-0.125, 0.125]$ . Secondly, the weights and bias of the output layer are initialized randomly uniformly in  $[-10^{-3}, 10^{-3}]$ . The two steps make sure that the outputs are near zero. Finally, we add 1 to the output layer so that the initial shaping weights of all state-action pairs are near 1.

All these networks are optimized using the Adam optimizer. The learning rates for optimizing the policy and the value function networks are  $10^{-4}$  and  $2 \times 10^{-4}$ , respectively. The learning rate for updating the potential network of the DPBA method is  $5 \times 10^{-4}$ . The learning rate for optimizing the shaping weight function of the BiPaRS methods is set differently in different tests. In the tests with the original shaping reward function, the learning rate is  $10^{-5}$ , while in the tests with the harmful shaping reward function (i.e., the first adaptability test) and the random shaping reward function (i.e., the third adaptability test), it is  $5 \times 10^{-4}$ . Recall that both the harmful and random shaping reward functions make it difficult to learn a good policy. We set the learning rate higher in the two tests to make the shaping weights change rapidly from the initial value 1.0. The discount rate  $\gamma$  is 0.999 in all tests. We summarize the above hyperparameter settings in Table 1. Note that the naive shaping (NS) method directly adds the shaping reward to the original reward function, so it has no other hyperparameters.

### D.2 MuJoCo

Recall that in the *MuJoCo* experiment, we adopt the shaping reward function  $f(s, a) = w(0.25 - \frac{1}{L} \sum_{i=1}^L |a_i|)$  to limit the average torque amount of the agent's action, where  $w$  is a task-specific weight which makes  $f(s, a)$  have the same scale as the true reward in the initial learning phase. Now we show the value of  $w$  in each task in Table 2.

Since the MuJoCo tasks are more complicated than the cartpole task, the policy and value function networks of the base learner PPO have three hidden layers with 64 units and *relu* activation in each layer. The RCPO algorithm also adopts PPO as the base learner and uses the same network architectures. The potential network of DPBA has two *tanh* hidden layers and each layer also has 64

Algorithm	Hyperparameters
Base Learner (PPO)	policy network: two 8-unit FC hidden layers, $relu$ activation value network: two 32-unit FC hidden layers, $relu$ activation threshold of probability ratio clipping ( $\epsilon$ ): 0.5 update period: 20,000 steps number of epoches per update: 50 batch size: 1024 GAE parameter ( $\lambda$ ): 0.95 optimizer: Adam learning rate of policy network: $10^{-4}$ learning rate of value network: $2 \times 10^{-4}$ discount rate ( $\gamma$ ): 0.999
DPBA	potential network: 16-unit FC layer + 8-unit FC layer, $tanh$ activation optimizer: Adam learning rate of potential network: $5 \times 10^{-4}$
BiPaRS	shaping weight network: 16-unit FC layer + 8-unit FC layer, $tanh$ activation optimizer: Adam learning rate of shaping weight network: $10^{-5}$ in the original test, and $5 \times 10^{-4}$ in the adaptability tests initial shaping weight: 1.0 shaping weight range: $(-\infty, +\infty)$

Table 1: The hyperparameters of the tested algorithms in the *cartpole* experiment

units. The shaping weight function network of our BiPaRS methods has two  $tanh$  hidden layers, with 16 and 8 units in the first and second layers, respectively.

All the tested algorithms perform model update every 20,000 training steps. One updating process contains 50 epochs and each updating epoch uses a batch of 1024 samples. The threshold for clipping the probability ratio is 0.2 in all of the five MuJoCo tasks. The discount rate  $\gamma$  is 0.999 and the parameter  $\lambda$  of GAE is 0.95. The Lagrange multiplier of RCPO is bounded in  $[0, 10000]$ . The neural network models of all algorithms are optimized by the Adam optimizer. The learning rates for optimizing the policy and value function networks are  $10^{-4}$  and  $2 \times 10^{-4}$ , respectively. The learning rates for updating the potential network of DPBA and the shaping weight function of BiPaRS-EM are  $5 \times 10^{-4}$ . The BiPaRS-MGL and BiPaRS-IMGL algorithms use a learning rate of  $10^{-3}$  to update their shaping weight functions. The RCPO algorithm uses a dynamic learning rate to update its Lagrange multiplier, which starts from  $5 \times 10^{-5}$  and exponentially decays with a factor  $1 - 10^{-9}$ .

One thing should be noted is that the *Humanoid-v2* task has a much higher state dimension than the other four tasks, which means that the neural networks of the tested algorithms have much more parameters. Therefore, for training the BiPaRS-IMGL algorithm in *Humanoid*, directly approximating the meta-gradient  $\nabla_\phi \theta'$  according to Equation (A.13) is extremely difficult because the Hessian matrix  $\nabla_\theta g_\theta(s_i, a_i)^\top$  with respect to the policy parameter  $\theta$  has  $\mathcal{O}(n^3)$  computational complexity. To address this issue, we make the policy network a two-layer network with only 32 units in each layer and ignore the term  $\alpha \sum_{i=1}^N \tilde{Q}(s_i, a_i) \nabla_\theta g_\theta(s_i, a_i)^\top$  in Equation (A.13). We summarize all the hyperparameter settings in the *MuJoCo* experiment in Table 3.

	<i>Swimmer-v2</i>	<i>Hopper-v2</i>	<i>Humanoid-v2</i>	<i>Walker2d-v2</i>	<i>HalfCheetah-v2</i>
<i>w</i>	20	16	20	10	100

Table 2: The task-specific weight  $w$  in each of five *MuJoCo* tasks

Assume that there are  $M$  monks and the total amount of the rice gruel is  $L$ . Without loss of generality, we assume that the rice gruel is assigned to the monks according to the order  $1, 2, \dots, M$ . The decision-making at each step is two-fold. Firstly, the system chooses one from the  $M$  monks. Secondly, the chosen monk is the assigner, who should determine the amount of rice gruel to the current monk to be assigned.

Formally, this can be modeled as follows. At each step  $t$  ( $t = 1, 2, \dots, M$ ), the system first chooses a monk  $m_t \in \{1, \dots, M\}$ . Then the chosen monk has to choose an action  $l_t \in [0, L - \sum_{t'=1}^{t-1} l_{t-1}]$ , which stands for the amount of rice gruel assigned to the monk at the assign order  $t$ . Therefore, the problem has only  $M + 1$  states and the state transitions are deterministic:

$$S_1 \rightarrow S_2 \rightarrow \dots \rightarrow S_M \rightarrow S_T, \quad (\text{C.25})$$

where  $S_T$  is the terminal state.

There are  $M + 1$  agents in the system, namely the  $M$  monks and the system. Importantly, we assume that each monk tries to maximize its own utility, while the system tries to make the utility of each monk the same. For any step  $t$ , let  $s_t$  denote the state and  $a_t = (m_t, l_t)$  denote the joint action of the system and the chosen monk. Then for any monk  $i \in \{1, \dots, M\}$ , the reward function is

$$R_i(s_t, a_t) = R_i(s_t, m_t, l_t) = \begin{cases} l_t & \text{if } i = t, \\ 0 & \text{o.w.,} \end{cases} \quad (\text{C.26})$$

which means that only the monk at the assign order  $t$  gets the rice gruel assigned by monk  $m_t$ . The system reward function is

$$R_0(s_t, a_t) = R_0(s_t, m_t, l_t) = -\left|\frac{L}{M} - l_t\right|, \quad (\text{C.27})$$

which stands for the absolute fairness of the system.

Algorithm	Hyperparameters
Base Learner (PPO)	<p>policy network: three 64-unit FC hidden layers,  <math>relu</math> activation</p> <p>value network: three 64-unit FC hidden layers,  <math>relu</math> activation</p> <p>threshold of probability ratio clipping (<math>\epsilon</math>): 0.2</p> <p>update period: 20,000 steps</p> <p>number of epoches per update: 50</p> <p>batch size: 1024</p> <p>GAE parameter (<math>\lambda</math>): 0.95</p> <p>optimizer: Adam</p> <p>learning rate of policy network: <math>10^{-4}</math></p> <p>learning rate of value network: <math>2 \times 10^{-4}</math></p> <p>policy gradient clip norm: 1.0</p> <p>value fuction gradient clip norm: 1.0</p> <p>discount rate (<math>\gamma</math>): 0.999</p>
DPBA	<p>potential network: two 64-unit FC layers,  <math>tanh</math> activation</p> <p>optimizer: Adam</p> <p>learning rate of potential network: <math>5 \times 10^{-4}</math></p> <p>potential network gradient clip norm: 10.0</p>
BiPaRS	<p>shaping weight network: 16-unit FC layer + 8-unit FC layer,  <math>tanh</math> activation</p> <p>optimizer: Adam</p> <p>learning rate of shaping weight network: <math>5 \times 10^{-4}</math> for BiPaRS-EM,  and <math>5 \times 10^{-4}</math> for BiPaRS-MGL and BiPaRS-IMGL</p> <p>shaping weight network gradient clip norm: 10.0</p> <p>initial shaping weight: 1.0</p> <p>shaping weight range: <math>[-1, 1]</math></p>
RCPO	<p>Lagrange multiplier lower bound: 0</p> <p>Lagrange multiplier upper bound: 10000</p> <p>initial learning rate of Lagrange multiplier: <math>5 \times 10^{-5}</math></p> <p>decay factor of learning rate: <math>1 - 10^{-9}</math></p>
Other	<p>policy network of BiPaRS-IMGL in <i>Humanoid-v2</i>:</p> <p>two 32-unit FC hidden layers, <math>relu</math> activation</p>

Table 3: The hyperparameters of the tested algorithms in the *MuJoCo* experiment