Neural Computing and Applications (2022) 34:1673–1686
https://doi.org/10.1007/s00521-021-06259-1

S.I. : ADAPTIVE AND LEARNING AGENTS 2020

Policy invariant explicit shaping: an efficient alternative to reward shaping

Paniz Behboudian[1,4] · Yash Satsangi[2] · Matthew E. Taylor[1,4] · Anna Harutyunyan[3] · Michael Bowling[1,4,5]

Received: 16 November 2020 / Accepted: 19 June 2021 / Published online: 28 September 2021

Abstract

Reinforcement learning (RL) is a powerful learning paradigm in which agents can learn to maximize sparse and delayed reward signals. Although RL has had many impressive successes in complex domains, learning can take hours, days, or even years of training data. A major challenge of contemporary RL research is to discover how to learn with less data. Previous work has shown that domain information can be successfully used to shape the reward; by adding additional reward information, the agent can learn with much less data. Furthermore, if the reward is constructed from a potential function, the optimal policy is guaranteed to be unaltered. While such *potential-based reward shaping* (PBRS) holds promise, it is limited by the need for a well-defined potential function. Ideally, we would like to be able to take arbitrary advice from a human or other agent and improve performance without affecting the optimal policy. The recently introduced *dynamic potential-based advice* (DPBA) was proposed to tackle this challenge by predicting the potential function values as part of the learning process. However, this article demonstrates theoretically and empirically that, while DPBA can facilitate learning with good advice, it does in fact alter the optimal policy. We further show that when adding the correction term to "fix" DPBA it no longer shows effective shaping with good advice. We then present a simple method called *policy invariant explicit shaping* (PIES) and show theoretically and empirically that PIES can use arbitrary advice, speed-up learning, and leave the optimal policy unchanged.

Keywords Artificial intelligence · Machine learning · Reinforcement learning · Dynamic potential-based reward shaping

✉ Paniz Behboudian
behboudi@ualberta.ca

Yash Satsangi
Y.Satsangi@tilburguniversity.edu

Matthew E. Taylor
matthew.e.taylor@ualberta.ca

Anna Harutyunyan
harutyunyan@google.com

Michael Bowling
mbowling@ualberta.ca

[1] Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada

[2] Department of Cognitive Science and Artificial Intelligence, Tilburg University, Tilburg, The Netherlands

[3] DeepMind, London, England

[4] Alberta Machine Intelligence Institute, Edmonton, Alberta, Canada

[5] DeepMind, Edmonton, Alberta, Canada

1 Introduction

An RL agent interacts with its surrounding environment by taking actions and receiving rewards and observations in return. The aim of the agent is to learn a policy (a mapping from states to actions) that maximizes a reward signal [26]. In many cases, the reward signal is sparse and delayed so that learning a good policy can take an excessively long time. For example, the Open AI Five agent [20] required 180 years worth of game experience per day of training; similarly, grand-master level StarCraft agent AlphaStar [27], required 16,000 matches as training data. One approach to accelerate learning is to add an external source of advice. The practice of providing an RL agent with additional rewards to improve learning is called *reward shaping* and the additional reward is called the shaping reward. *Shaping* was proposed by the psychologist B. F. Skinner for animal training analogous to a sculptor shaping clay to form the final statue: a complex task can be

learned faster by first solving its simpler approximates [7, 24, 26]. Shaping the *reward* in RL is meant to facilitate learning by reinforcing the intermediate behaviours related to achieving the main goal. One of the oldest examples of applying reward shaping is the work by Gullapalli and Barto [7] where a simulated robot hand was trained to press a key via learning a set of successive approximations to the original task. However, naively augmenting the original reward function with shaping can alter the optimal policy of the RL agent [22]. For example, Randløv and Alstrøm [22] showed that adding a shaping reward (that at first seems reasonable) could "distract" the agent. The authors utilized reward shaping to accelerate their RL agent trying to learn how to ride a bicycle. When they provided positive reinforcement for making transitions toward the goal, the agent was misguided to find a loop as the optimal behaviour, accumulating positive rewards over and over:

> "In our first experiments we rewarded the agent for driving towards the goal but did not punish it for driving away from it. Consequently the agent drove in circles with a radius of 20-50 meters around the starting point. Such behavior was actually rewarded by the reinforcement function..."

Potential-based reward shaping (PBRS) [18, 19, 29, 30] is a well-known solution that addresses the policy-variance problem and allows an RL agent to incorporate external advice without altering its optimal policy by deriving the shaping reward from a potential function. Given a static potential function, PBRS defines the shaping reward as the difference in the potentials of states (or state-action pairs) when an agent makes a transition from one state to another. Ng et al. [18] showed that PBRS is guaranteed to be policy invariant: using PBRS does not alter the optimal policy.

While PBRS achieves policy invariance, it may be difficult or impossible for a person or agent to express their advice as a potential-based function. Instead, it would be preferable to allow the use of more direct or intuitive advice in the form of an arbitrary function; e.g., giving a positive/negative feedback for a desirable/undesirable actions. The ideal reward shaping method then would have three properties:

1. Be able to use an arbitrary reward function as advice,
2. Keep the optimal policy unchanged in the presence of the additional advice, and
3. Improve the speed of learning of the RL agent when the advice is good.[1]

[1] We used "good" and "bad" in simple relative terms. We refer to advice as "good" to simply mean that one would expect that it would help the speed of learning, e.g., it rewards optimal actions more often than non-optimal actions.

Harutyunyan et al. [8] attempted to tackle the same problem by proposing the framework of *dynamic potential-based advice* (DPBA), where the idea is to dynamically learn a potential function from arbitrary advice, which can then be used to define the shaping reward. Importantly, the authors claimed that if the potential function is initialized to zero then DPBA is guaranteed to be policy invariant. We show in this work that this claim is not true, and hence, the approach is unfortunately *not* policy invariant. We confirm our finding theoretically and empirically. We then propose a fix to the method by deriving a correction term, and show that the result is theoretically sound, and empirically policy-invariant. However, our empirical analysis shows that the *corrected* DPBA fails to accelerate the learning of an RL agent provided with useful advice.

We introduce a simple algorithm, *policy invariant explicit shaping* (PIES), and show that PIES can allow for arbitrary advice, is policy invariant, and can accelerate the learning of an RL agent. PIES biases the agent's policy toward the advice by presenting a new hyper-parameter to control the amount of advice influence on the agent policy: having more bias at the start of the learning, when the agent is the most in need of guidance and over time, gradually decaying the bias to zero, which ensures policy invariance. Several experiments confirm that PIES ensures convergence to the optimal policy when the advice is misleading and also accelerates learning when the advice is useful.

Concretely, this article makes the following contributions:

1. Identifies an important flaw in a published reward shaping method.
2. Verifies the flaw exists, empirically and theoretically by showing that the optimal policy can be altered by advice.
3. Provides a correction to the method, but empirically shows that it introduces additional complications, where good advice no longer improves learning speed.
4. Introduces and verifies a simple approach that achieves the goals of the original method.

The rest of the paper is outlined as this: first we will provide some background material about RL and reward shaping, then we narrow down to DPBA as a specific type of reward shaping and demonstrate some experimental results (with good advice) which are central to our contributions, then in the next section we study the problem of changing the optimal policy with DPBA in depth, and contrast the (bad advice) findings with the previous section; lastly we introduce PIES, our alternative that satisfies all of the goals we have set, and justify how it can overcome the aforementioned drawbacks of previous methods.

2 Background

A *Markov Decision Process* (MDP) [21], is described by the tuple $\langle S, A, T, \gamma, R \rangle$. At each time step, the environment is in a state $s \in S$, the agent takes an action $a \in A$, and the environment transitions to a new state $s' \in S$, according to the transition probabilities $T(s, a, s') = \Pr(s'|s, a)$. Additionally, the agent (at each time step) receives a reward for taking action a in state s according to the reward function $R(s, a)$. Finally, γ is the discount factor, specifying how to trade off future rewards and current rewards.

A deterministic policy π is a mapping from states to actions, $\pi : S \rightarrow A$, that is, for each state, s, $\pi(s)$ returns an action, $a = \pi(s)$. The *state-action value function* $Q^{\pi}(s, a)$ is defined as the expected sum of discounted rewards the agent will get if it takes action a in state s and follows the policy π thereafter.

$$Q^{\pi}(s, a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R(s_{t+k}, a_{t+k}) \Big| s_t = s, a_t = a, \pi\right].$$

The agent aims to find the optimal policy denoted by π^* that maximizes the expected sum of discounted rewards, and the state-action value function associated with π^* is called the optimal state-action value function, denoted by $Q^*(s, a)$:

$$Q^*(s, a) = \max_{\pi \in \prod} Q^{\pi}(s, a),$$

where $\prod$ is the space of all policies.

The action value function for a given policy π satisfies the *Bellman equation*:

$$Q^{\pi}(s, a) = R(s, a) + \gamma \mathbb{E}_{s', a'}[Q^{\pi}(s', a')],$$

where s' is the state at the next time step and a' is the action the agent takes on the next time step, and this is true for all policies.

The Bellman equation for the optimal policy π^* is called the *Bellman optimality equation*:

$$Q^*(s, a) = R(s, a) + \gamma \mathbb{E}_{s', a'}[Q^*(s', a')].$$

Given the optimal value function $Q^*(s, a)$, the agent can retrieve the optimal policy by acting greedily with respect to the optimal value function:

$$\pi^*(s) = \arg\max_{a \in A} Q^*(s, a).$$

The idea behind many reinforcement learning algorithms is to learn the optimal value function Q^* iteratively. For example, Sarsa [26] learns Q-values with the following update rule, at each time step t (Q_0 can be initialized arbitrarily):

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t \delta_t, \quad (1)$$

where

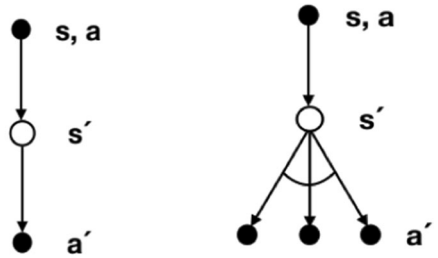
$$\delta_t = R(s_t, a_t) + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)$$

is the *temporal-difference* error (TD-error), s_t and a_t denotes the state and action at time step t, Q_t denotes the estimate of Q^* at time step t, and α_t is the learning rate at time step t. The TD-error implies that the agent bootstraps from its current estimation of the value function for computing the target (the temporally successive estimates) in the error term. If all state-action pairs continue to be visited (for infinite number of times), with an appropriate learning rate and a bounded reward, these Q estimates are guaranteed to converge to Q^* for all s, a, and the policy converges to π^* [26, 28]. Q-learning is another RL approach to estimate Q^* with a similar update rule to Sarsa but with a δ_t:

$$\delta_t = R(s_t, a_t) + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t),$$

where s_t and a_t denotes the state and action at time step t, Q_t denotes the estimate of Q^* at time step t, and α_t is the learning rate at time step t. Q-learning belongs to the family of *off-policy* algorithms: the target policy for which the optimal value function is being learned differs from the behaviour policy which the agents follows to collect the learning samples. Here, the target policy is greedy while the behaviour policy could be some different policy such as ϵ-greedy (which will be discussed shortly). The backup diagrams for Sarsa(0) and Q-learning are shown in Fig. 1a, b respectively, to illustrate the basis of their updates.

While learning iteratively, a good policy should dedicate some time for *exploration* to discover new states and actions as well as some time to *exploit* the knowledge it already gained—just like what humans do, e.g., ordering a new dish in a restaurant or rather choosing from those that have been already tried, with maximum deliciousness. One such policy is ϵ-greedy under which the probability of choosing a random action is ϵ and the greedy action is $1 - \epsilon$.



(a) Sarsa(0) **(b)** Q-learning

Fig. 1 The backup diagrams indicating the update rule behind (**a**) Sarsa (0) and (**b**) Q-learning algorithms.

2.1 Potential-based reward shaping

In cases where the rewards are sparse, reward shaping can help the agent learn faster by providing an additional *shaping reward* F. However, the addition of an arbitrary reward can alter the optimal policy of a given MDP [22]. *Potential-based reward shaping* (PBRS) addresses the problem of adding a shaping reward function F to an existing MDP reward function R, without changing the optimal policy by defining F to be the difference in the *potential* of the current state s and the next state s' [18]. Specifically, PBRS restricts the shaping reward to the following form: $F(s,s') := \gamma\Phi(s') - \Phi(s)$, where $\Phi : S \to \mathbb{R}$ is the potential function. Ng et al. [18] showed that expressing F as the difference of potentials is the sufficient condition for the agent to be *policy invariant*. That is, if the original MDP $\langle S, A, T, \gamma, R\rangle$ is denoted by M and the shaped MDP $\langle S, A, T, \gamma, R + F\rangle$ is denoted by M' (M' is same as M but offers the agent an extra reward F in addition to R) then the optimal value function of M and M' for any state-action pair (s, a) satisfies:

$$Q^*_{M'}(s,a) = Q^*_M(s,a) - \Phi(s)$$

where Φ is the *bias term*. Given $Q^*_{M'}$, the optimal policy π^* can simply be obtained by adding the bias term as:

$$\pi^*(s) = \arg\max_{a\in A} Q^*_M(s,a) = \arg\max_{a\in A}(Q^*_{M'}(s,a) + \Phi(s)).$$

Because the bias term only depends on the agent's state, the optimal policy of the shaped MDP M', does not differ from that of the original MDP M. To also include the shaping reward on actions, Wiewiora et al. [30] extended the definition of F to state-action pairs by defining F to be: $F(s,a,s',a') := \gamma\Phi(s',a') - \Phi(s,a)$, where Φ is dependent on both the state and the action of the agent. Now the bias term is also dependent on the action taken at state s, therefore the agent must follow the policy

$$\pi^*(s) = \arg\max_{a\in A}(Q^*_{M'}(s,a) + \Phi(s,a))$$

in order to be policy-invariant.

In the work by Wiewiora [29], potential-based reward shaping with static potentials (which do not change with time), proved to be equivalent to initializing the value function with the potential function, given that the learning algorithm is using a tabular temporal difference method with an advantage-based exploration policy and the same sequence of experience during learning. Thus far, none of the methods that we discussed admits an arbitrary form of advice, hence fulfills the ideal shaping goals provided in Sect. 1. In the next section, we further explain some approaches with potentials that changes dynamically with time and through empirical evaluation we show whether dynamic potentials can be harnessed to achieve our shaping goals.

3 Dynamic potential-based reward shaping

PBRS, as discussed in Sect. 2.1, is restricted to external advice that can be expressed in terms of a potential function. Therefore, it does not satisfy the first of the three goals; i.e., it cannot admit arbitrary advice. Finding a potential function Φ that accurately captures the advice can be challenging. To allow an expert to specify an arbitrary function R^{expert} and still maintain all the properties of PBRS one might consider *dynamic PBRS*. In this section, we explain dynamic PBRS in details and show experimental results to evaluate based on our three criteria.

Dynamic PBRS uses a potential function Φ_t that can be altered online to form the dynamic shaping reward F_t, where subscript t indicates the time over which F and Φ vary. Devlin and Kudenko [5] used dynamic PBRS as $F_{t+1}(s,s') := \gamma\Phi_{t+1}(s') - \Phi_t(s)$, where t and $t+1$ are the times that the agent arrives at states s and s', respectively. They derived the same guarantees of policy invariance for dynamic PBRS as static PBRS. To admit an arbitrary reward, Harutyunyan et al. [8] suggested learning a dynamic potential function Φ_t given the external advice in the form of an arbitrary bounded function, R^{expert}. To do so, the following method named *dynamic potential-based advice* (DPBA) is proposed: define $R^{\Phi} := -R^{expert}$, and learn a *secondary value function* Φ via the following update rule at each time step:

$$\Phi_{t+1}(s,a) := \Phi_t(s,a) + \beta\delta^{\Phi}_t \tag{2}$$

where $\Phi_t(s,a)$ is the current estimate of Φ, β is the Φ function's learning rate, and

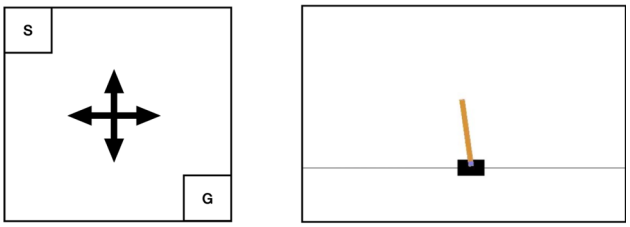
$$\delta^{\Phi}_t := R^{\Phi}(s,a) + \gamma\Phi_t(s',a') - \Phi_t(s,a)$$

is the Φ function's TD error. All the while, the agent learns the Q values using Sarsa (i.e., according to Eq. 1). In addition to the original reward $R(s, a)$ the agent receives a shaping reward given as:

$$F_{t+1}(s,a,s',a') := \gamma\Phi_{t+1}(s',a') - \Phi_t(s,a), \tag{3}$$

that is, the difference between the consecutively updated values of Φ.

Harutyunyan et al. [8] suggested that with this form of reward shaping, $Q^*_M(s,a) = Q^*_{M'}(s,a) + \Phi_0(s,a)$ for every s and a and therefore to obtain the optimal policy π^*, the agent should be greedy with respect to $Q^*_{M'}(s,a) + \Phi_0(s,a)$ by the following rule:



(a) Grid-World (b) Cart-Pole

Fig. 2 The two (**a**) grid-world and (**b**) cart-pole domains that DPBA was empericaly evaluated on them.

$$\pi^*(s) = \arg\max_{a \in A}(Q^*_{M'}(s,a) + \Phi_0(s,a)), \tag{4}$$

and thus if $\Phi_0(s,a)$ is initialized to zero, the above biased policy in Eq. 4, reduces to the original greedy policy:

$$\pi^*(s) = \arg\max_{a \in A} Q^*_{M'}(s,a) = \arg\max_{a \in A} Q^*_{M}(s,a).$$

As this approach is central to our contribution, we give a more thorough treatment of this work. DPBA was empirically evaluated on two episodic tasks: a 20×20 grid-world and a cart-pole problem. In the grid-world experiment as shown in Fig. 2a, the agent starts each episode from the top left corner until it reaches the goal located at the bottom right corner, within a maximum of 10,000 steps. The agent can move along the four cardinal directions and the state is the agent's coordinates (x, y). The reward function is $+1$ upon arrival at the goal state and 0 elsewhere. The advice, R^{expert}, for any state action is:

$$R^{\text{expert}}(s,a) := \begin{cases} +1 & \text{if } a \text{ is right or down} \\ 0 & \text{otherwise} \end{cases}.$$

This article replicates the same grid-world environment in our later experiments.

In the cart-pole task [16] as shown in Fig. 2b, the goal is to balance a pole on top of a cart as long as possible. The cart can move along a track and each episode starts in the middle of the track with the pole upright. There are two possible actions: applying a force of $+1$ or -1 to the cart. The state consists of a four dimensional continuous vector, indicating the angle and the angular velocity of the pole, and the position and the velocity of the cart. An episode ends when the pole has been balanced for 200 steps or the pole falls, and the reward function encourages the agent to balance the pole.

To replicate this experiment[2], this article used the OpenAI Gym [2] implementation (CartPole-v0).We should note that,there are slight differences between our implementation of cart-pole and the version used in the DPBA paper [8], making the results not directly comparable. In that paper, 1) if the cart attempts to move beyond the ends of the track, the cart bounces back, and 2) there is a negative reward if the pole drops and otherwise the reward is zero. In contrast, in OpenAI Gym, 1) if the cart moves beyond the track's boundaries, the episode terminates, and 2) the reward function is $+1$ on every step the pole is balanced and 0 if the pole falls. The advice for this task is defined as:

$$R^{expert}(s,a) := o(s,a) \times c,$$

where $o : S \times A \rightarrow \{0,1\}$ is a function that triggers when the pole direction is aligned with the force applied to the cart (i.e., when the cart moves in the same direction as the pole is leaning towards, the agent is rewarded). We set $c = 0.1$.

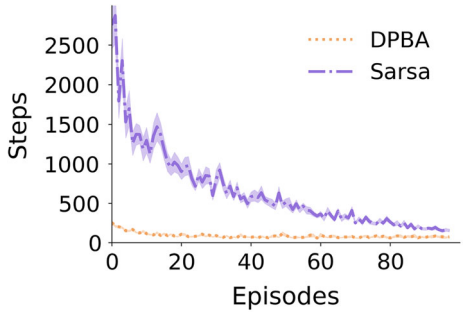
Figure 3 shows the performance of the DPBA method, compared to a simple Sarsa learner not receiving any expert advice, in the grid-world and the cart-pole domains. We used the same set of hyper-parameters as used in [8] for the grid-world. For learning the cart-pole task, the agent used a linear function approximation for estimating the value function via Sarsa(λ) and tile-coded feature representation [25] with the implementation from the open-source software (publicly available at Richard Suttons's website). The weights for Q and Φ were initialized uniformly random between 0 and 0.001. For tile-coding, we used 8 tilings, each with 2^4 tiles (2 for each dimension). We used a wrapping tile for the angle of the pole for a more accurate state-representation. With a wrapping tile one can generalize over a range (e.g. $[0, 2\pi]$) rather than stretching the tile to infinity, and then wrap-around. λ was set to 0.9 and γ to 1. For learning rates of Q and Φ value functions, α and β, we swept over the values [0.001, 0.002, 0.01, 0.1, 0.2]. The best parameter values according to the area under curve (AUC) of each line for Fig. 3b is reported in Table 1.

These results agree with the prior work, showing that the agent using the DPBA method learned faster with this good advice, relative to not using the advice (i.e. the DPBA line is converging faster to the optimal behaviour). Note that in the grid-world task the desired behaviour is to reach the goal as fast as possible. Consequently, for this task the lower is better in plots (such as the ones in Fig. 3) showing steps (y-axis) versus episodes (x-axis). In contrast, in cart-

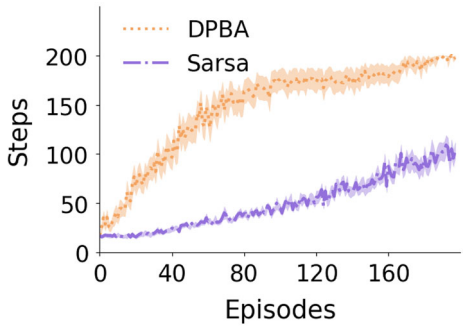
Table 1 Parameters values for Fig. 3b

Agent	α	β
Sarsa	0.1	–
DPBA	0.02	0.1

[2] To assist with reproducibility, the implementations of the algorithms used in this article's experiments are accessible at https://github.com/panizbehboudian/Useful-Policy-Invariant-Shaping-from-Arbitrary-Advice.



(a) Grid-World (Lower is better.)



(b) Cart-Pole (Higher is better.)

Fig. 3 The y-axis shows the number of time steps taken to finish each episode (on x-axis) averaged over (**a**) 50 and (**b**) 30 runs. The shaped agent with DPBA is compared with a Sarsa learner without shaping in (**a**) grid-world and (**b**) cart-pole domains. Shaded areas correspond to the standard error. Parameter values for figure (**b**) are reported in Table 1

pole since the goal is to keep the pole balanced longer, the higher line which indicates more steps, is better. The results in Fig. 3 show that DPBA method satisfies criterion 1 (it can use arbitrary rewards) and criterion 3 (good advice can improve performance). However, as we argue in the next section, a flaw in the proof of the original paper means that criterion 2 is not satisfied: the optimal policy can change, i.e., advice can cause the agent to converge to a sub-optimal policy. This was not empirically tested in the original paper and thus this flaw was not noticed.

4 DPBA can affect the optimal policy

The previous section described DPBA, a method that can incorporate an arbitrary expert's advice in a reinforcement learning framework by learning a potential function Φ iteratively and concurrently with the shaped state-action values, $Q_{M'}$. In this section, we argue that DPBA does not satisfy policy-invariance property, the second desired criterion we listed for a successful reward shaping method. We back our argument with theoretical proof and experimental results, and further we demonstrate how DPBA behaves with good and bad advice, when we attempt to fix the flaw.

4.1 Theoretical proof

Harutyunyan et al. [8] claimed that if the initial values of Φ, Φ_0, are initialized to zero, then the agent can simply follow a policy that is greedy with respect to $Q_{M'}$ to achieve policy invariance. In this section, we show that this claim is unfortunately not true: initializing $\Phi_0(s,a)$ to zero is not sufficient to guarantee policy invariance.

To prove our claim, we start by defining terms. We will compare Q-value estimates for a given policy π in two MDPs, the original MDP denoted by M described by the tuple $\langle S, A, T, \gamma, R\rangle$, and the MDP that is shaped by DPBA, M', described by the tuple $\langle S, A, T, \gamma, R + F_{t+1}\rangle$, where $F_{t+1}(s,a,s',a') = \gamma\Phi_{t+1}(s',a') - \Phi_t(s,a)$.

Let $R'_{t+1} := R + F_{t+1}$, given a policy π, at any time step t, $Q^{\pi}_{M'}(s,a)$ can be defined as:

$$= \mathbb{E}\left[\sum_{k=0}^{\infty}\gamma^k R'_{t+k+1}(s_{t+k}, a_{t+k}, s_{t+k+1}, a_{t+k+1})\middle| \begin{matrix} s_t = s, \\ a_t = a \end{matrix}\right].$$

By writing R' in terms of R and F:

$$= \mathbb{E}\left[\sum_{k=0}^{\infty}\gamma^k \left(R(s_{t+k}, aa_{t+k}) + F_{t+k+1}(s_{t+k}, a_{t+k}, s_{t+k+1}, a_{t+k+1})\right)\middle| \begin{matrix} s_t = s, \\ a_t = a \end{matrix}\right].$$

The first term in the above expression (after separating the expectation) is the value function for the original MDP M for policy π.

$$= \underbrace{\mathbb{E}\left[\sum_{k=0}^{\infty}\gamma^k R(s_{t+k}, a_{t+k})\middle| \begin{matrix} s_t = s, \\ a_t = a \end{matrix}\right]}_{=Q^{\pi}_{M}(s,a)} + \mathbb{E}\left[\sum_{k=0}^{\infty}\gamma^k F_{t+k+1}(s_{t+k}, a_{t+k}, s_{t+k+1}, a_{t+k+1})\middle| \begin{matrix} s_t = s, \\ a_t = a \end{matrix}\right] \tag{5}$$

The second term in Eq. 5 can be expanded based on Eq. 3 as follows:

$$\mathbb{E}\left[\sum_{k=0}^{\infty}\gamma^k F_{t+k+1}(s_{t+k},a_{t+k},s_{t+k+1},a_{t+k+1})\middle|\begin{matrix}s_t=s,\\a_t=a\end{matrix}\right]$$
$$=\mathbb{E}\left[\sum_{k=0}^{\infty}\left(\gamma^{k+1}\Phi_{t+k+1}(s_{t+k+1},a_{t+k+1})\right.\right.$$
$$\left.\left.-\gamma^k\Phi_{t+k}(s_{t+k},a_{t+k})\right)\middle|\begin{matrix}s_t=s,\\a_t=a\end{matrix}\right]. \tag{6}$$

The two terms inside the infinite summation look quite similar, motivating us to rewrite one of them by shifting its summation variable k. This shift will let identical terms be cancelled out. However, we need to be careful. First, we rewrite the sums in their limit form. An infinite sum can be written as:

$$\sum_{i=i_0}^{\infty}x_i:=\lim_{W\to\infty}\sum_{i=i_0}^{W}x_i.$$

Using this definition, in Eq. 6 we can shift the first term's iteration variable as:

$$=\mathbb{E}\left[\lim_{W\to\infty}\left(\sum_{k=1}^{W+1}\gamma^k\Phi_{t+k}(s_{t+k},a_{t+k})\right.\right.$$
$$\left.\left.-\sum_{k=0}^{W}\gamma^k\Phi_{t+k}(s_{t+k},a_{t+k})\right)\middle|\begin{matrix}s_t=s,\\a_t=a\end{matrix}\right] \tag{7}$$
$$=\mathbb{E}\left[\lim_{W\to\infty}\left(\gamma^{W+1}\Phi_{t+W+1}(s_{t+W+1},a_{t+W+1})\right.\right.$$
$$\left.\left.-\gamma^0\Phi_t(s_t,a_t)\right)\middle|\begin{matrix}s_t=s,\\a_t=a\end{matrix}\right].$$

In Eq. 7, if $\Phi_t^{\pi}(s,a)$ is bounded, then the first term inside the limit will go to 0 as W approaches infinity. Note that this term will go to zero only for infinite horizon MDPs. In practice, it is common to assume a finite horizon MDP with a terminal state, in such cases, this extra term will remain and must be removed, for example, by defining the potential of the terminal state to be zero. The second term inside the limit does not depend on W and can be pulled outside the limit:

$$\mathbb{E}\left[\sum_{k=0}^{\infty}\gamma^k F_{t+k+1}(s_{t+k},a_{t+k},s_{t+k+1},a_{t+k+1})\middle|\begin{matrix}s_t=s,\\a_t=a\end{matrix}\right]$$
$$=\mathbb{E}\left[-\Phi_t(s_t,a_t)\middle|\begin{matrix}s_t=s,\\a_t=a\end{matrix}\right]=-\Phi_t(s,a) \tag{8}$$

Back to Eq. 5, if we apply Eq. 8, we will have:

$$Q_{M'}^{\pi}(s,a)=Q_M^{\pi}(s,a)-\Phi_t(s,a),$$

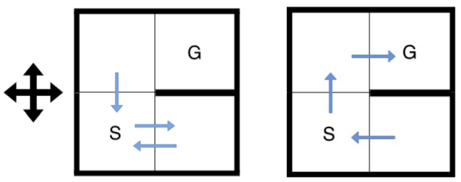
for all π (given that $s_t=s$ and $a_t=a$). Thus, for an agent to retrieve the optimal policy π_M^* given $Q_{M'}^*(s,a)$, it must act greedily with respect to $Q_{M'}^*(s,a)+\Phi_t(s,a)$:

$$\pi_M^*(s)=\arg\max_{a\in A}\left(Q_{M'}^*(s,a)+\Phi_t(s,a)\right). \tag{9}$$

Equation 9 differs from Eq. 4 (corresponding to Equation 17 in Harutyunyan et al. [8]) in that the bias term is Φ_t and not Φ_0. In other words, at every time step the Q values of the agent are biased by the *current estimate* of the potential function and not by the initial value of the potential function. The derived relation in Equation 17 of Harutyunyan et al. [8] is only valid for the first state-action pair that the agent visits (at $t=0$). For the rest of the time steps, it is not accurate to use the first time step's bias term to compensate the shaped value function. Thus, the zero initialization of Φ is not a sufficient condition for the agent to recover the true Q values of the original MDP, and it cannot be used to retrieve the optimal policy of the original MDP.

4.2 Empirical validation: unhelpful advice

We empirically validate the result above with a set of experiments. First, consider a deterministic 2×2 grid-world which we refer to it as the *toy example*, depicted in Fig. 4. The agent starts each episode from state S and can move in the four cardinal directions (as depicted in the figure) until it reaches the goal state G (within a maximum of 100 steps). Moving towards a wall (indicated by bold lines), causes no change in the agent's position. The reward is 0 on every transition except the one ending in the goal state, resulting in a reward of $+1$ and episode termination. For advice, we assume that the "expert" rewards the agent for making transitions away from the goal. Blue arrows inside the grid in Fig. 4a represent the expert advised state-transitions. The agent receives a $+1$ from the expert by executing the advised transitions. Because this advice is encouraging poor behavior, we expect that it would slow down the learning (rather than accelerate it), but if a shaping method is policy invariant, the agent should still eventually converge to the optimal policy.



(a) Bad expert **(b)** Good expert

Fig. 4 The toy example domain with advised transitions indicated by blue arrows. The expert gives a reward of $+1$ if the agent takes an advised transition and 0 otherwise. The bad expert in (**a**) tries to keep the agent away from the goal while the good expert in (**b**) rewards transitions towards the goal.

Table 2 Parameters values for Figs. 5 and 6

Agent	α	β
Sarsa	0.05	–
DPBA, good advice	0.2	0.5
DPBA, bad advice	0.2	0.5
corrected DPBA, good advice	0.2	0.1
corrected DPBA, bad advice	0.05	0.2

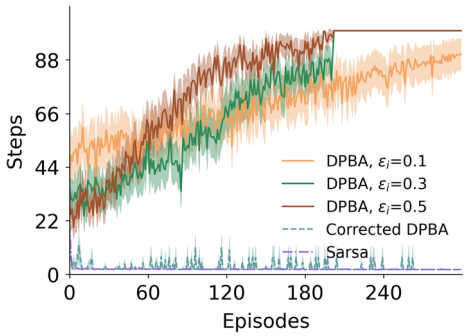


Fig. 5 The *y*-axis shows number of time steps taken to finish each episode with the bad expert. The shaped agent with corrected DPBA (with different initial ϵ values) is compared with the shaped agent with DPBA and an unshaped Sarsa agent. Shaded areas correspond to the standard error averaged over 50 runs (lower is better). Parameter values used for generating this figure are reported in Table 2

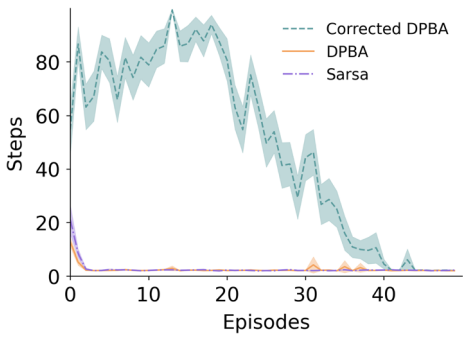


Fig. 6 The *y*-axis shows number of time steps taken to finish each episode with the good expert. The shaped agent with corrected DPBA is compared with the shaped agent with DPBA and an unshaped Sarsa agent. Shaded areas correspond to the standard error averaged over 50 runs (Lower is better). Parameter values used for generating this figure are reported in Table 2

The learner uses Sarsa(0) to estimate Q values with $\gamma = 0.3$. We ran the experiment for both the learner with *corrected DPBA*, Eq. 9, and the learner with *DPBA*, Eq. 4, using an ϵ-greedy policy. Φ and Q were initialized to 0 and ϵ was decayed from 0.1 to 0. For the learning rates of the Q and Φ value functions, α and β, we swept over the values [0.05, 0.1, 0.2, 0.5]. The best values according to the AUC of the each line are reported in Table 2.

Figure 5 depicts the length of each episode as the number of steps taken to finish the episode. The *Sarsa* line indicates the learning curve for a Sarsa(0) agent without shaping. Figure 5 shows that DPBA does not converge to the optimal policy with a bad advice. This figure also confirms our result that Φ_t (and not Φ_0 as proposed in Harutyunyan et al. [8]) is the sufficient correction term to recover the optimal policy for maximizing the MDP's original reward.

Finally, we verify that this is not simply an artefact of the agent exploiting too soon, and repeat the same experiments for different exploration rates, ϵ. We considered two more different values for initial exploration rate, ϵ_i: 0.3 and 0.5. The corresponding lines in Figure zfig:toyspsexamplespscorrectedspsbad confirm additional exploration does not let DPBA obtain the optimal policy. Figure 5 also confirms that the corrected policy as derived in Eq. 9 converges to the optimal policy even when the expert advice is bad.

4.3 Empirical validation: helpful advice

The previous section showed that DPBA is not a policy invariant shaping method since initializing the values of Φ to zero is not a sufficient condition for policy invariance. We showed that DPBA can be corrected by adding the correct bias term and indeed policy invariant. While the addition of the correct bias term guarantees policy invariance, we still need to test our third criterion for the desired reward shaping algorithm — does corrected DPBA accelerate the learning of a shaped agent with good expert advice?

Figure 6 shows the results for repeating the same experiment as the previous subsection but with the good expert which is shown in Fig. 4b (i.e., from each state the expert encourages the agent to move towards the goal). Here, since the expert is encouraging the agent to move towards the goal, we expect the shaped agent to learn faster than the agent that is not receiving a shaping reward. However, Fig. 6 shows that the corrected agent does not learn faster with good advice. To our surprise, the advice actually slowed down the learning, even though the corrected DPBA agent did eventually discover the optimal policy, as expected.

To explain the corrected DPBA behaviour, one needs to closely look at how the Q and Φ estimates are changing and how they interact with each other. The corrected DPBA adds the latest value of Φ for selecting the next action at each time step, in order to correct the shaped Q value; however, the Φ value which has been used earlier to shape the reward function and updating the Q value might be different than the latest value. This difference is specially

Table 3 Φ_t values at different episodes from the toy example experiment with good advice (Fig. 6)

Episodes	10	20	30	40	50	60
$\Phi(1, right)$	− 0.09	− 0.37	− 0.56	− 0.79	− 0.92	− 0.97
$\Phi(2, up)$	− 0.15	− 0.45	− 0.65	− 0.95	− − 1.14	− 1.22
$\Phi(3, left)$	− 0.29	− 0.84	− 0.97	− 1.00	− 1.01	− 1.01

more significant in the earlier episodes when δ^{Φ} and δ^{Q} have higher values, before these value functions stabilize. Let us examine this behaviour through the toy example experiment by looking at the actual numeric values of Φ from the simulation. Assume the case that Φ and Q has been initialized to zero and since the advice is always a positive signal for advised state-action pairs, their Φ values have to be negative. With such a Φ the latest Φ values are more negative than the earlier values used for shaping the reward as shown in Table 3, which in fact discourages the desired behaviour. Almost after episode 50, Φ values start to stabilize (note that the changes in values are negligible compared to earlier episodes) and that is when the corrected DPBA starts finding the optimal policy and converging to the same point as Sarsa.

While the corrected DPBA guarantees policy invariance, it fails in satisfying the third goal for ideal reward shaping (i.e., speed up learning of an agent with a helpful advice).

The main conclusion of this section is that none of the mentioned reward shaping methods for incorporating expert advice satisfies the three ideal goals. DPBA [8] can lead to faster learning if the expert offers good advice but it is not policy invariant. The corrected DPBA proposed in this section is provably policy invariant but it can lead to slower learning even when provided with good advice.

5 Policy invariant explicit shaping

In this section, we introduce the *policy invariant explicit shaping* (PIES) algorithm that satisfies all of the goals we specified in Sect. 1 for reward shaping. PIES is a simple algorithm that admits arbitrary advice, is policy invariant, and speeds up the learning of the agent depending on the expert advice.

Unlike potential-based reward shaping, the main idea behind PIES is to use the expert advice explicitly without modifying the original reward function. Not changing the reward function is the principal feature that both simplifies PIES and makes analysing how it works easier. The PIES agent learns the original value function Q_M as in Eq. 1, while concurrently learning a secondary value function Φ based on expert advice as in Eq. 2. To exploit the arbitrary advice, we introduce a new hyper-parameter ξ which controls that to what extent the agent's current behaviour is biased towards the expert's advice. As mentioned, PIES does not change the value function Q but rather, only during action selection to bias the agent's policy explicitly, it makes a decision based on a combination of Q_M and $-\Phi$ weighted by ξ_t at each time step t. To bias the agent more in early learning stages, ξ starts with a higher value and to enforce the policy to be relied on Q and ensuring policy invariance, it decays gradually to zero. For example, for a Sarsa(0) agent equipped with PIES, when the agent wants to act greedily, it will pick the action that maximizes $Q_t(s,a) - \xi_t \Phi_t(s,a)$ at each time step. The optimal policy would be:

$$\pi^*_M(s) = \arg\max_{a \in A} \big(Q^*_M(s,a) - \xi_t \Phi_t(s,a)\big),$$

where ξ_t decays to 0 before the learning terminates. We add the negation of Φ to shape the Q values since Φ is accumulating the $-R^{expert}$. We kept the Φ reward function, R^{Φ}, the same as DPBA for the sake of consistency. However, one can easily revert the sign to set $R^{\Phi} = R^{expert}$ and add Φ to Q to shape the agent with PIES.

Decaying ξ_t to 0 over time removes the effect of shaping, guaranteeing that the agent will converge to the optimal policy, making PIES policy-invariant. More specifically, the update rule for Q_t remains the same, our policy is a GLIE policy [23], and we do on-policy updates; therefore, Q converges to Q^* under the same conditions for Sarsa. The speed of decaying ξ determines how long the advice will continue to influence the agent's learned policy. Choosing the decay speed of ξ can be related to how beneficial it is to utilize the advice and can be done in many different ways. For this article, we only decrease ξ at the end of each episode with a linear regime. More specifically, the value of ξ during episode e is:

$$\xi_e := \begin{cases} \xi_{e-1} - \dfrac{1}{C} & \text{if } \xi_{e-1} \text{is not } 0 \\ 0 & \text{otherwise} \end{cases}, \xi_1 = 1, \tag{10}$$

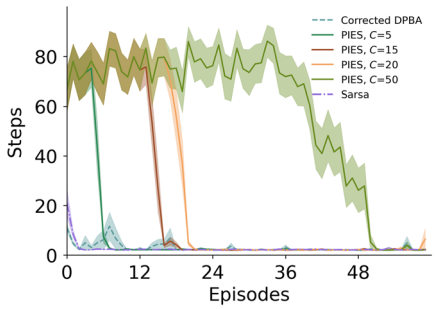
where C is a constant that determines how fast ξ will be decayed; i.e., the greater C is the slower the bias decays. The complete pseudo-code is shown in Algorithm.

Algorithm 1 PIES with Sarsa(0) updates for Q, Φ

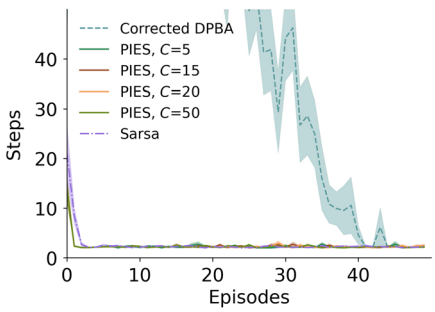
Algorithm parameters: step sizes α for Q and β for Φ, small $\epsilon > 0$, decay factor ξ
Initialize $Q(s,a)$ and $\Phi(s,a)$ for all $s \in \mathbf{S}$, $a \in \mathbf{A}$ arbitrarily except that $Q(terminal,.) = 0$, $\Phi(terminal,.) = 0$, $\xi = 1$
for each episode **do**
    Initialize S
    Choose A from S using policy derived from $Q + \xi\Phi$ (e.g., ϵ-greedy)
    for each step of episode until S is terminal **do**:
        Take action A, observe R, R^{expert}, S'
        Choose A' from S' using policy derived from $Q+\xi\Phi$ (e.g., ϵ-greedy)
        if $\xi_t > 0$ **then**
            $\Phi(S,A) \leftarrow \Phi(S,A) + \beta[R^{expert} + \gamma\Phi(S',A') - \Phi(S,A)]$
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
        $S \leftarrow S'; A \leftarrow A';$
    $\xi \leftarrow next\_\xi(\xi)$

We first demonstrate empirically that PIES fulfills all three goals in the toy example. We then show how it performs against previous methods in the grid-world and the cart-pole problems (which were originally tested for DPBA), when provided with good advice. All the domains specifications are the same as before.

The plot of the agents' performance in the toy example in Fig. 7 shows learning curves of the corrected DPBA, PIES, and the Sarsa learner. Figure 7a is for the bad expert shown in Fig. 4a and b is for the good expert as in Fig. 4b. Sarsa (0) was used to estimate state-action values with $\gamma = 0.3$ and an ϵ-greedy policy. Φ and Q were initialized to 0 and ϵ decayed from 0.1 to 0. For learning rates of Q and Φ value functions, α and β, we swept over the values [0.05, 0.1, 0.2, 0.5]. We show the performance of PIES for different C values of [5, 10, 20, 50]. It is worth mentioning that for setting the decaying speed of ξ (through setting C) one should consider the quality of the advice; i.e., a smaller C for a relatively bad advice as it decays the effect of the adversarial bias faster and a larger C for a good advice to keep the bias longer. The best values of C (according to the AUC of each line) along with learning parameters are reported in Table 4.



(a) Advice from the bad expert (Lower is better.)



(b) Advice from the good expert (Lower is better.)

Fig. 7 The y-axis shows the number of steps taken to finish each episode in the toy example. The figures compare PIES with the corrected DPBA and a Sarsa learner without shaping when the advice is (**a**) bad and (**b**) good. Shaded areas correspond to the standard error over 50 runs. The parameter values of this figure are reported in Table 4

As expected, with PIES the agent was able to find the optimal policy even with the bad expert. In the case of the beneficial advice, PIES enabled the agent to learn the task faster. The speed-up, though, is not remarkable in the toy problem, as the simple learner is also able to learn in very few episodes. Furthermore, with good advice the sensitivity of the performance to the decay speed was almost negligible. With bad advice, even with a C value as high as 50 (which is a poor choice and incompatible with the advice), the PIES agent was able to converge to the optimal policy soon after the bias effect vanishes which implies robustness over long-persisting adversarial advice.

Table 4 Parameters values for Fig. 7

Agent	α	β	C
Sarsa	0.05	–	–
corrected DPBA, good advice	0.2	0.1	–
corrected DPBA, bad advice	0.05	0.2	–
PIES, good advice	0.05	0.2	50
PIES, bad advice	0.1	0.2	5

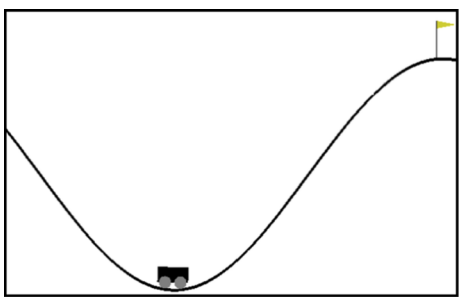


Fig. 8 The mountain car domain where a car is trying to reach the goal marked at top of the right hill

Figure 9 better demonstrates how PIES boosts the performance of the agent learning with good advice in three more complex tasks: the grid-world and the cart-pole domains which were described in Sect. 3 and the mountain car task. For grid-world and cart-pole tasks similar learning parameters (those that we do not re-state) inherited their values from previous experiments. To find the best C, values of [50, 100, 200, 300] were swept. In the grid-world task (Fig. 9a) α and β were selected over the values of [0.05, 0.1, 0.2, 0.5]. In the cart-pole task (Fig. 9b), for setting the learning rates, the values of [0.001, 0.002, 0.01, 0.1, 0.2] were swept.

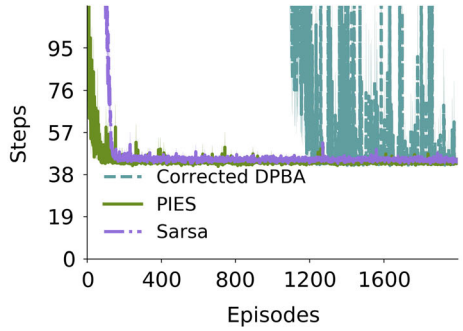
In the mountain car task [17] the agent's goal is to drive the car up to the top of a hill at right as shown in Fig. 8. The challenge is that since the car's engine is not strong enough, the car cannot reach the goal with full throttle in one pass. The solution is to drive back to the left hill to build up the momentum needed for climbing the steep slope and then accelerate toward the goal. The car moves along a one-dimensional track with three possible actions: accelerate to the left, right or cease any acceleration. The states are indicated by the car position and velocity at every time-step. Each episode starts with the car placed at a random position in between the two hills with zero velocity and terminates when either the goal is reached or after 10,000 time-steps. The reward is -1 for every step that the car has not reached the goal location and 0 when the goal is reached. We used Open-AI MountainCar-v0 implementation for this experiment [2].

The advice for this task is defined as:

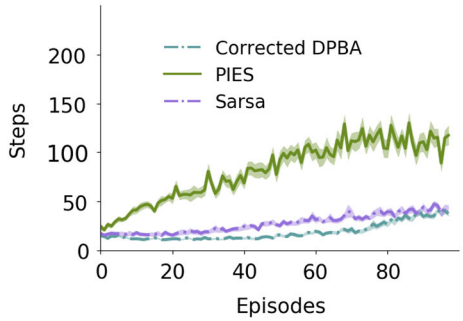
$$R^{expert}(s,a) := \begin{cases} +1 & \textit{if } a \textit{ is } \text{accelerate to right} \\ & \textit{and velocity is positive} \\ +1 & \textit{if } a \textit{ is } \text{accelerate to left} \\ & \textit{and velocity is negative} \\ 0 & \textit{otherwise} \end{cases}.$$

For learning the mountain car task, the agent used a linear function approximation for estimating the value function via Sarsa(λ) and tile-coded feature representation. The weights for Q and Φ were initialized uniformly random between 0 and 0.001. For tile-coding, we used 8 tilings, each with 8^2 tiles (8 for each dimension). λ was set to 0.9 and γ to 1. For learning rates of Q and Φ value functions, α and β, we swept over the values [0.1, 0.2, 0.5, 0.6]. To find the best C, values of [200, 300, 400, 500] were swept.

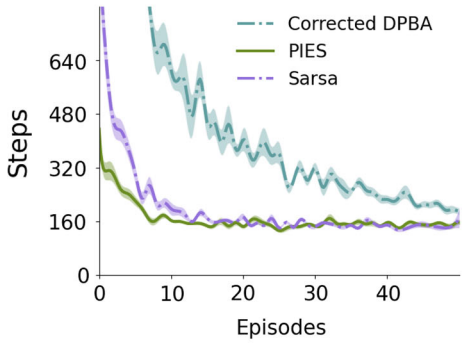
As before, the best parameter values according to the AUC of each line for Fig. 9a–c are reported in Tables 5, 6, and 7 respectively. Just like before, for the cart-pole task's plot the upper lines indicate better performance whereas for the grid-world and the mountain car tasks the lower lines are better.



(a) Grid-world (Lower is better.)



(b) Cart-pole (Higher is better.)



(c) Mountain Car (Lower is better.)

Fig. 9 The length of each episode as the number of steps in (**a**) the grid-world, (**b**) the cart-pole, and (**c**) the mountain car domains. The plot depicts PIES versus the corrected DPBA and a Sarsa learner without shaping. Shaded areas correspond to the standard error over (**a**) 50, (**b**) 30, and (**c**) 20 runs. Parameter values for figure (**a**), (**b**), and (**c**) is reported in Tables 5, 6, and 7, respectively.

Table 5 Parameters values for Fig. 9a

Agent	α	β	C
Sarsa	0.05	-	-
corrected DPBA	0.1	0.01	-
PIES	0.05	0.5	100

Table 6 Parameters values for Fig. 9b

Agent	α	β	C
Sarsa	0.1	–	–
corrected DPBA	0.02	0.1	–
PIES	0.2	0.5	200

Table 7 Parameters values for Fig. 9c

Agent	α	β	C
Sarsa	0.5	–	–
corrected DPBA	0.5	0.5	–
Pies	0.5	0.5	500

PIES correctly used the good advice in all the three domains and improved learning over the Sarsa learner, without changing the optimal policy (i.e., PIES approached the optimal behaviour with a higher speed compared to the Sarsa learner). PIES performed better than the corrected DPBA as expected, since the corrected DPBA is not capable of accelerating the learner with good advice. PIES is a reliable alternative for DPBA when we have an arbitrary form of advice, regardless of the quality of the advice. As shown, PIES satisfies all three desired criteria.

6 Related work

PIES is designed to overcome the flaws we exposed with DBPA and thus tackles the same problem as DPBA. Hence, PIES could be compared with methods such as TAMER [9–13] and heuristically accelerated Q-learning (HAQL) [1]. While TAMER and HAQL have similar goals, they differ on how advice is incorporated. PIES and DBPA use external advice by learning a *value function* online. TAMER, on the other hand, fits a model to the advice in a supervised fashion. In the earlier work of TAMER [9–11], the agent is provided with the interactive human advice, H, and tries to model it as $\hat{H}$ with supervised learning. Then it acts such that it maximizes the expected immediate modeled reward, i.e., $\hat{H}$. In the later TAMER+RL [12] variant, the agent also takes the environmental reward into consideration. For combining the MDP's reward with the pre-trained human reinforcements model, Knox and Stone [12] proposed several combination techniques. In some of these techniques, the agent augments either the reward or value function with $\hat{H}$ which are the most relevant methods to PIES: the reward augmentation method augments the reward function with $\hat{H}$, the value augmentation method augments the value function, and the action biasing method augments the value function only during the action selection (similar to adding Φ to Q in PIES). In all of the mentioned methods, $\hat{H}$ is weighted by an annealing factor, decaying over time (which plays a similar role as β in PIES). While TAMER+RL uses a pre-learned model for the human reward (which is learned offline), PIES does not need a prior phase of interacting with the environment for approximating Φ. Moreover, $\hat{H}$ is estimating the immediate human advice, ignoring the long-term effect of actions, while PIES estimates the expected return of the advice. We should mention that Knox and Stone [12] used PBRS in one their methods by substituting the static potential function with $\max_a H(s,a)$. Since this potential function is static and depends only on states, this method is not relevant to PIES. Interestingly, the results from TAMER+RL established that action biasing method, which resembles PIES the most, performed the best amongst all of the techniques investigated.

Further extending TAMER+RL, Knox and Stone [13] later provided a more thorough empirical study on the selected most successful methods and also adjusted the framework to learn from human advice and the MDP's reward simultaneously (unlike the previous work that learns $\hat{H}$ beforehand); however, the framework remains myopic and limited to immediate reward. Once again, their empirical results confirmed that action biasing was the best method (along with the action control method) among the top-selected methods. They discuss that for incorporating human advice, it is best to affect the exploration directly (by changing action selection only) rather than modifying the reward signal and/or the value function. By incorporating the advice only for action selection (like the way action biasing, control sharing, and also PIES do), the agent learn the accurate values based on its own experience and can safely achieve the goal.

Similar to TAMER, HAQL biases the policy with a heuristic function that dynamically changes with time through an update rule, based on the current estimate of Q value function. HAQL is identical to TAMER's action biasing, if the heuristic function is substituted with a learned model of the advice. Bianchi et al. [1] also mention a weight variable (analogous to ξ) to influence the effect of the heuristic on the policy. Although PIES uses a similar approach to bias the policy toward the advice, instead of a heuristic function, PIES learns a value function from the advice (independent of the Q value function). The authors showed that under certain assumptions, HAQL's Q value function converges to the optimal value function and thus it preserves the optimal policy. However, there was no

explicit discussion on the role of the heuristic function weight variable for policy invariance.

While TAMER and HAQL do not restrict the form of external advice, we argue that incorporating the advice in form of a value function as PIES does is more general. This is similar to the case of standard RL, where we work to maximize total discounted future rewards instead of acting myopically only based on the immediate reward. Unlike other methods, PIES emphasizes the role of the decay factor which makes PIES policy invariant. With PIES, the agent explicitly reasons about long-term consequences of following external advice, and that successfully accelerates learning, particularly in the initial (and most critical) steps.

Another line of work, related to multi-objectivization of RL by reward shaping [3, 4], uses multiple potential functions through PBRS. This research is orthogonal to PIES, but could be combined with it if multiple sources of advice are present.

Other approaches [6, 7, 14] share some commonality with PIES and DBPA in that they try to approximate a good potential function; for example by solving an abstract MDP and computing its final value function [15], or by learning a more generalized value function of the MDP with state aggregation [6]. However, they do not tackle the challenge of incorporating arbitrary external advice.

7 Conclusion and discussion

This article exposed a flaw in DPBA, a previously published algorithm with the aim of shaping the reward function with an arbitrary advice without changing the optimal policy. We used empirical and theoretical arguments to show that it is not policy invariant, a key criterion for reward shaping. Further, we derived the corrected DPBA algorithm with a corrected bias component. However, based on our empirical results the corrected algorithm fails to improve learning when leveraging useful advice resulting in a failure to satisfy the speed-up criterion. To overcome these problems, we proposed a simple approach, called PIES. We show theoretically and empirically that it guarantees the convergence to the optimal policy for the original MDP, agnostic to the quality of the arbitrary advice while it successfully speeds up learning from a good advice. Therefore, PIES satisfies all of the goals for ideal shaping.

Central to PIES is the decaying factor, ξ, with which one can tune the effect of bias (Φ) during learning. For the purpose of this paper, we used a simple speed decay by setting a hyper-parameter called C, but this might not be the most efficient approach. Finding a good schedule for ξ opens up a new problem for future work. More specifically, an idea to explore could be learning ξ as a new parameter with respect to the quality of the advice or even based on the effectiveness of advice in certain areas of the state space. Besides, another path to explore could be applying PIES in more complex tasks with deep RL and nonlinear function approximation to demonstrate the method scalability.

Acknowledgements Part of this work has taken place in the Intelligent Robot Learning (IRL) Lab at the University of Alberta, which is supported in part by research grants from the Alberta Machine Intelligence Institute; a Canada CIFAR AI Chair, Amii; CIFAR, Compute Canada; and NSERC.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Bianchi RA, Ribeiro CH, Costa AH (2004) Heuristically accelerated Q–learning: a new approach to speed up reinforcement learning. In: Brazilian symposium on artificial intelligence. Springer, pp. 245–254
2. Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J, Zaremba W(2016) OpenAI Gym. arXiv preprint arXiv:1606.01540
3. Brys T, Harutyunyan A, Vrancx P, Nowé A, Taylor ME (2017) Multi-objectivization and ensembles of shapings in reinforcement learning. Neurocomputing 263:48–59
4. Brys T, Harutyunyan A, Vrancx P, Taylor ME, Kudenko D, Nowé A (2014) Multi-objectivization of reinforcement learning problems by reward shaping. In: 2014 international joint conference on neural networks. IEEE, pp. 2315–2322
5. Devlin SM, Kudenko D (2012) Dynamic potential-based reward shaping. In: Proceedings of the 11th international conference on autonomous agents and multiagent systems, pp. 433–440
6. Grześ M, Kudenko D (2010) Online learning of shaping rewards in reinforcement learning. Neural Netw 23(4):541–550
7. Gullapalli V, Barto AG (1992) Shaping as a method for accelerating reinforcement learning. In: Proceedings of the 1992 IEEE international symposium on intelligent control, pp. 554–559
8. Harutyunyan A, Devlin S, Vrancx P, Nowé A (2015) Expressing arbitrary reward functions as potential-based advice. In: The association for the advancement of artificial intelligence, pp. 2652–2658
9. Knox WB, Fasel IR, Stone P (2009) Design principles for creating human-shapable agents. In: The association for the

advancement of artificial intelligence spring symposium: agents that learn from human teachers, pp. 79–86
10. Knox WB, Stone P (2008) TAMER: Training an agent manually via evaluative reinforcement. In: 2008 7th IEEE international conference on development and learning, pp. 292–297
11. Knox WB, Stone P (2009) Interactively shaping agents via human reinforcement: the TAMER framework. In: Proceedings of the 5th international conference on knowledge capture. ACM, pp. 9–16
12. Knox WB, Stone P (2010) Combining manual feedback with subsequent MDP reward signals for reinforcement learning. In: Proceedings of the 9th international conference on autonomous agents and multiagent systems: Vol. 1, pp. 5–12
13. Knox WB, Stone P (2012) Reinforcement learning from simultaneous human and MDP reward. In: Proceedings of the 11th international conference on autonomous agents and multiagent systems-vol. 1, pp. 475–482
14. Marom O, Rosman B (2018) Belief Reward Shaping in Reinforcement Learning. Proc AAAI Conf Artif Intell 32(1)
15. Marthi B (2007) Automatic shaping and decomposition of reward functions. In: Proceedings of the 24th international conference on machine learning. ACM, pp. 601–608
16. Michie D, Chambers RA (1968) Boxes: an experiment in adaptive control. Mach Intell 2(2):137–152
17. Moore A (2002) Efficient memory-based learning for robot control
18. Ng AY, Harada D, Russell S (1999) Policy invariance under reward transformations: theory and application to reward shaping. Int Conf Mach Learn 99:278–287
19. Ng AY, Jordan MI (2003) Shaping and policy search in reinforcement learning. Ph.D. thesis, University of California, Berkeley
20. OpenAI (2018) OpenAI Five. https://blog.openai.com/openai-five/
21. Puterman ML (2014) Markov decision processes: discrete stochastic dynamic programming. Wiley, New Jersey
22. Randløv J, Alstrøm P (1998) Learning to drive a bicycle using reinforcement learning and shaping. Int Conf Mach Learn 98:463–471
23. Singh S, Jaakkola T, Littman ML, Szepesvári C (2000) Convergence results for single-step on-policy reinforcement-learning algorithms. Mach Learn 38(3):287–308
24. Skinner BF (1958) Reinforcement today. Am Psychol 13(3):94
25. Sutton RS (1996) Generalization in reinforcement learning: successful examples using sparse coarse coding. In: Advances in neural information processing systems, pp. 1038–1044
26. Sutton RS, Barto AG (2018) Reinforcement learning: an introduction, 2nd edn. The MIT Press. http://incompleteideas.net/book/the-book-2nd.html
27. Vinyals O, Babuschkin I, Czarnecki WM, Mathieu M, Dudzik A, Chung J, Choi DH, Powell R, Ewalds T, Georgiev P, Oh J, Horgan D, Kroiss M, Danihelka I, Huang A, Sifre L, Cai T, Agapiou JP, Jaderberg M, Vezhnevets AS, Leblond R, Pohlen T, Dalibard V, Budden D, Sulsky Y, Molloy J, Paine TL, Gulcehre C, Wang Z, Pfaff T, Wu Y, Ring R, Yogatama D, Wünsch D, McKinney K, Smith O, Schaul T, Lillicrap T, Kavukcuoglu K, Hassabis D, Apps C, Silver D (2019) Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature 575:350–354
28. Watkins CJ, Dayan P (1992) Q-learning. Mach Learn 8(3–4):279–292
29. Wiewiora E (2003) Potential-based shaping and Q-value initialization are equivalent. J Artif Intell Res 19:205–208
30. Wiewiora E, Cottrell GW, Elkan C (2003) Principled methods for advising reinforcement learning agents. In: Proceedings of the 20th international conference on machine learning, pp. 792–799