Orientation-Preserving Rewards' Balancing in Reinforcement Learning

Jinsheng Ren, Shangqi Guo, and Feng Chen, *Member, IEEE*

***Abstract*— Auxiliary rewards are widely used in complex reinforcement learning tasks. However, previous work can hardly avoid the interference of auxiliary rewards on pursuing the main rewards, which leads to the destruction of the optimal policy. Thus, it is challenging but essential to balance the main and auxiliary rewards. In this article, we explicitly formulate the problem of rewards' balancing as searching for a Pareto optimal solution, with the overall objective of preserving the policy's optimization orientation for the main rewards (i.e., the policy driven by the balanced rewards is consistent with the policy driven by the main rewards). To this end, we propose a variant Pareto and show that it can effectively guide the policy search toward more main rewards. Furthermore, we establish an iterative learning framework for rewards' balancing and theoretically analyze its convergence and time complexity. Experiments in both discrete (grid word) and continuous (Doom) environments demonstrated that our algorithm can effectively balance rewards, and achieve remarkable performance compared with those RLs with heuristically designed rewards. In the ViZDoom platform, our algorithm can learn expert-level policies.**

***Index Terms*— Automatic rewards' balancing, auxiliary rewards, Pareto solutions, reinforcement learning, reward design.**

I. Introduction

AUXILIARY rewards are widely used in complex reinforcement learning tasks [1]–[4]. Many studies [5]–[8] have shown that it can effectively alleviate the problem of low sample efficiency and improve the speed of policy learning. For example, in some tasks in Doom (a first-person video game) [9], the main objective is to kill as many enemies as possible. If only a single reward for the main objective is provided, which we call the main reward, it is difficult to drive policy learning due to the long-term credit assignment problem [5]. To this end, auxiliary rewards are often designed for some other objectives, such as navigating through a map, collecting medkits and clips, which play an indispensable role in the optimization of the main objective. However, the optimality of policy learning relies on the balancing between main and auxiliary rewards, which still remains a key challenge [6].

Manuscript received April 8, 2020; revised November 30, 2020; accepted May 11, 2021. This work was supported in part by the National Natural Science Foundation of China under Grant 61671266 and Grant 61836004, and in part by the Tsinghua-Guoqiang Research Program under Grant 2019GQG006. *(Corresponding author: Feng Chen.)*

The authors are with the Department of Automation, Tsinghua University, Beijing 100086, China, also with the Beijing Innovation Center for Future Chips, Beijing 100084, China, and also with the LSBDPA Beijing Key Laboratory, Beijing 100084, China (e-mail: rjs17@mails.tsinghua.edu.cn; gsq15@mails.tsinghua.edu.cn; chenfeng@mail.tsinghua.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2021.3080521.

Digital Object Identifier 10.1109/TNNLS.2021.3080521

There are mainly two ways of using auxiliary rewards in order to make them balanced with the main rewards. One way is to design an auxiliary reward for each other objective and combine them with the main rewards into a scalarization reward function based on prior knowledge [10], [11]. This makes agents have the ability to learn the relationship among objectives by putting more or less emphasis on each reward. However, once the prior is unreliable, agents are often misled to collect auxiliary rewards excessively so that they violate the purpose of tasks [12]. Another way attempts to separate each reward and studies how to switch among them to implement the desired behavior [8], [13], [14]. Although this way avoids the combination issues, designing switching rules are also required for both RL expertise and domain-specific knowledge. In a word, neither of the above two paradigms provides a principled method to avoid the interference of auxiliary rewards on pursuing the main rewards, which leads to the destruction of the optimal policy.

In this article, our goal is to automatically balance rewards through preserving the policy's optimization orientation for the main rewards (i.e., the policy driven by the balanced rewards is consistent with the policy driven by the main rewards). To make progress on this problem, we should keep in mind that, searching for more main rewards is the central purpose of policy learning, while the auxiliary rewards merely boost this process. Inspired by multiobjective optimization [15], [16], we show that this process is similar to the search for a Pareto optimal solution, where the main rewards on this Pareto solution are not dominated by that on any other Pareto solutions. However, previous Pareto-based algorithms [17]–[19] are not directly applicable to rewards' balancing since they could induce many Pareto solutions that are not expected. We, therefore, propose a variant Pareto to evaluate trajectories by incorporating preference information for the main rewards into the original Pareto [16], so as to preserve the search orientation for the main rewards. In this way, rewards are balanced by fitting the evaluation, and the policy is optimized by maximizing the balanced rewards.

Formally, we establish an iterative learning framework called orientation-preserving reinforcement learning (OP-RL), which sequentially performs rewards' balancing and policy optimization. This aims to gradually infer the correct relationship among rewards. In this way, one of the most concerned issues is whether rewards can be stably and correctly balanced as iterations go on. To this end, two sufficient conditions for convergence of OP-RL are given by abstracting it as a Markov chain. Furthermore, we analyze the boundary of OP-RL's time complexity in theory.

We summarize our main contributions as follows.

1) We model the problem of rewards' balancing as searching for a Pareto optimal solution and propose a variant Pareto for preserving the optimization orientation for the main rewards.
2) We establish OP-RL, an iterative learning framework for rewards' balancing, and theoretically analyze its convergence and time complexity.
3) We show in various experiments that OP-RL can effectively balance rewards and achieve remarkable performance compared with those RLs with heuristically designed rewards. In the ViZDoom platform [9], our algorithm can learn expert-level policies.

II. Related Work

Even though adding auxiliary rewards has been powerful in many experiments, it can be also misleading when the auxiliary rewards are used improperly. Therefore, it is of great significance to develop algorithms that can balance main and auxiliary rewards. In terms of access to the reward function, three related paradigms are described as follows.

A. Hand-Design Rewards' Balancing

Reward shaping [6], [20] is a popular paradigm that provides the agent with shaping reward (closely related to auxiliary reward) for improving performance rather than only providing main rewards when the goal has been achieved. Even though shaping rewards have been powerful in many experiments [21], [22], they are typically combined with the main rewards by the user based on their knowledge of the domain [12]. The work in [6] proposed a theory for the optimal-preserved problem: if a shaping reward is potential-based, the optimal policy of the task is preserved. However, one problem is that detailed knowledge of the potential of states is not often available [23]. Therefore, this theory can hardly guide rewards balance in practice.

To avoid the combination issues, Li and Czarnecki [13] and Gábor *et al.* [24] studied how to switch among rewards to drive policy learning. They design the switching rule via an ordering or a preference relation among multiple rewards to optimize the main rewards while putting constraints on other auxiliary rewards. However, the design of an appropriate lexicographic ordering of all rewards and their constraint criteria still requires much prior knowledge of the problem domain.

In a word, neither of the above two paradigms provides a general and principled method to balance rewards, which easily leads to the destruction of the optimal policy. In our work, we propose an algorithm that can balance rewards in a self-supervised manner and theoretically prove its optimal policy is preserved. This greatly improves the ability of RL algorithms to solve complex tasks.

B. Learn the Reward Function

Inverse reinforcement learning (IRL) [25] can be applied to infer the reward function from demonstrations. Unfortunately, it is difficult to provide high-quality demonstrations in complex tasks due to the ambiguous relationship among multiple objectives [26]. This even results in suboptimal behavior when the demonstrator is suboptimal [27]. Compared with these works, our work has no demonstration but only a Pareto-based trajectory evaluator. In this way, we can balance rewards by fitting the evaluation. This setting allows our work to be adopted in tasks where demonstrations are not available.

Preference-based reinforcement learning (PBRL) [26], [28] is a paradigm that learns from experts' preferences rather than handy designs of numeric rewards. It uses preferences between states, actions, or trajectories as supervised signals, aiming to find a policy that is maximally consistent with them. Although our work follows some basic ideas in PBRL, there is an essential difference as following. Instead of eliciting preferences on trajectories by experts throughout the whole learning process, we propose a variant Pareto to preserve the policy's searching orientation for the main rewards. This can not only save lots of labor costs but also improve the efficiency of solving the optimization problem.

C. Pareto Optimality in Multiobjective RL

Since our work adopts the Pareto concept, we refer readers to [15] and [29] for a complete overview of multiobjective reinforcement learning (MORL). MORL is a paradigm that requires an agent to optimize two or more objectives at the same time. Since objectives may be contradictory, there is no single optimal policy. In terms of the number of learned policies, MORL algorithms can be divided into two classes. One class is to optimize a scalar value, including the weighted sum method [16], [30], the ranking-based method [13], [24], and so on. They aim to find an expert-satisfactory policy, which represents the preferences among the multiple objectives as specified by a user or derived from the problem domain [31], [32]. Another class is to use Pareto domination to constitute the Pareto front and find a set of policies that approximate the Pareto front [17]–[19]. The main difference among multiple-policy approaches is the approximation scheme for the Pareto front.

However, these Pareto-based algorithms are not directly applicable to rewards' balancing. They could induce many Pareto solutions that are not expected due to treating each solution equally. We, therefore, propose a variant Pareto that incorporates preference information for the main rewards into the original Pareto. It aims to obtain a Pareto optimal solution, in which the main rewards on this solution are not dominated by that of any other Pareto solutions.

III. Preliminaries

A. Markov Decision Process

Reinforcement learning [33] is a paradigm that allows an agent to optimize its policy by interacting with a given environment. The agent is rewarded or punished for its behavior,

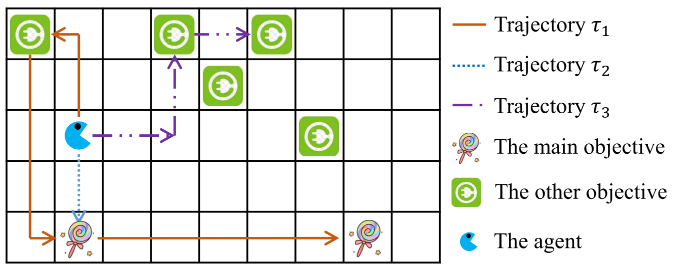


Fig. 1. Purpose of the agent is to collect as many candies as possible by weighing the importance of collecting and charging. τ_1 is the optimal trajectory that reasonably balances collection and charging. τ_2 and τ_3 are suboptimal trajectories since both are biased toward one of the objectives, respectively.

and the aim is to maximize the accumulated reward over time. A popular model for such problems is Markov decision processes (MDPs), defined by a set of states S, a set of actions A, a transition function $T(s_{t+1}|s_t, a_t)$ that maps the state s_t and action a_t to a probability over all possible next states s_{t+1}, and a reward function R that maps each state s_t and action a_t taken in it to a numeric reward $r_t = R(s_t, a_t)$. The discount factor, γ, defines how important future rewards are. We define a full trajectory τ as a sequence of transitions from initial state s_0 to a final state s_n; $\tau = \{s_0, a_0, r_0, \ldots, s_n\}$. We define return $g(\tau)$ to be a function of a full trajectory $\tau = \{s_0, a_0, r_0, \ldots, s_n\}$. The nature of the return is determined by the discount factor γ. The return accumulates rewards over the sequence $g(\tau) = r_0+\gamma r_1+\cdots+\gamma^n r_n$. The goal of the agent is to find a policy π that maximizes the expected cumulative discounted return $\mathcal{J}^\pi$

$$\mathcal{J}^\pi = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]. \tag{1}$$

B. Problem Statement for Rewards' Balancing

In this section, we will elaborate on the problem of rewards' balancing through a simple but common example. Meanwhile, we answer the following two questions: 1) why the main rewards need to be balanced in addition to auxiliary rewards and 2) why we use dynamic weights to balance rewards.

Here, we focus on a type of complex task, in which the main difficulty lies in how to make an appropriate balance between multiple objectives, so as to achieve the main objective better. Fig. 1 shows a schematic describing the task. In this 5×9 grid world, the incomplete circle represents the agent, whose main objective is to collect as many candies as possible. Considering that the realization of the main objective cannot be separated from the help of other objectives, we assume that the agent has to be charged in time to make up for the power consumed by any action. In this way, τ_1 is the optimal trajectory that reasonably balances collection and charging. τ_2 and τ_3 are suboptimal trajectories since both are biased toward one of the objectives, respectively.

For the first question, let us consider the case where only the main rewards r_m exist. As shown in Fig. 1, whether the agent can collect a candy depends on its remaining power. In this way, the main rewards should be set as a function of the remaining power. The challenge lies in the fact that experts usually cannot get the exact form of the function, especially in complex tasks. In extreme case, let us consider that the agent could attain a reward whenever it collects a candy. Unfortunately, this setting can easily lead to a myopic policy as the following analysis. As shown in Fig. 1, we have $g(\tau_1) = (\gamma^8 + \gamma^{14}) * r_m$, and $g(\tau_2) = \gamma^2 * r_m$. This means, if $\gamma \leq 0.92$, then $g(\tau_1) < g(\tau_2)$. According to the objective function [see (1)], the system would prefer trajectory τ_2 to trajectory τ_1, even if trajectory τ_1 is actually better. This phenomenon is called short-sightedness [34], where the agent is more inclined to get short-term returns while ignoring the possibility of getting greater long-term returns. In theory, a larger γ can be used to alleviate this problem. However, in practice, this would probably cause difficulties in algorithm convergence [35]. Instead of selecting γ carefully, we address this difficulty by learning to balance the main rewards.

For the second question, let us consider the case where auxiliary rewards are set for other objectives. In this way, the reward is not a scalar value but a vector

$$\vec{r}(s, a) = (r_m(s, a), r_{a_1}(s, a), \ldots, r_{a_n}(s, a)) \tag{2}$$

where $r_{a_i}, i \in \{1, 2, \ldots, n\}$ represents the auxiliary reward for ith other objective. Since these objectives may conflict with each other, any policy can only realize a tradeoff among them. In our work, the weighted sum method $\vec{\omega}\cdot\vec{r}(s, a)$ is adopted to balance multiple rewards, which is one of the most common methods in MORL [16], [36]. The weight vector expresses the relative importance of each objective. However, the fixed weights' setting is not suitable for solving complex tasks. As shown in Fig. 1, charging stations are densely distributed in the environment. Even the weights for this objective are small enough to highlight the importance of collecting candy, this setting may still lead to unexpected policies that are heavily biased toward these other objectives, such as trajectory τ_3. In fact, the relative importance of each objective is related to the state of the agent. Thus, in our work, the dynamic weight vector is used to balance each reward and guide agents to learn the correct temporal relationship among these objectives.

IV. Orientation-Preserving Rewards' Balancing

Our goal is to balance rewards automatically through preserving the policy's optimization orientation for the main rewards and ensure that the policy driven by the balanced rewards is optimality-preserved. In the following, we model rewards' balancing as multiobjective optimization. We first introduce a variant Pareto called orientation-preserving Pareto (OP-Pareto) and then give an iterative optimization framework for reward balancing. Furthermore, we theoretically analyze the convergence and time complexity of the framework.

A. Rewards' Balancing as Multiobjective Optimization

We use the definition of the reward vector, as stated in Section III-B: $\vec{r}_t = [r_m(s_t, a_t), r_{a_1}(s_t, a_t), \ldots, r_{a_n}(s_t, a_t)]$, where m and $a_i (i \in \{1, 2, \ldots, n\})$ represent the main objective and the ith other objective, respectively. We use the weighted sum approach to combine each reward function, in which the weight vector $\vec{w}$ represents the relative importance among the multiple objectives. Besides, we extend $\vec{w}$ as a function of state–action pairs to express the time-variant importance. Our goal is to optimize $\vec{w}(s_t, a_t)$ so that the policy optimized by the weighted sum of reward function is consistent with that optimized only by the reward function from the main objective

$$\pi_w : \max E\left(\sum_t \sum_i w_i(s_t, a_t) \cdot r_i(s_t, a_t)\right) = \pi_m : \max E\left(\sum_t r_m(s_t, a_t)\right) \quad (3)$$

where $i \in \{m, a_1, a_2, \ldots, a_n\}$. π_w is the optimal policy optimized by the weighted sum of reward function, and π_m is the optimal policy optimized only by the main rewards.

According to (3), the goal of optimizing $\vec{w}(s_t, a_t)$ is to make the better the trajectory, the larger the cumulative return under the multiple reward function. Thus, the key problem is how to evaluate the trajectory. Based on the formulation of RL, the scalar cumulative return, which combines all rewards, serves as the role of evaluating behaviors. However, this usually misleads the agent to collect auxiliary rewards excessively without satisfying the purpose of the task. Intuitively, we can try to evaluate trajectories based on the accumulated main rewards that indicate the accomplishment of the main objective. However, we argue that this evaluation alone is not sufficient since it does not take into account the promotion effect of auxiliary rewards on policy learning. For example, although two players currently kill the same number of enemies, we believe that the player who has more health or clips is more capable of killing more enemies in the future. To this end, we model the problem of rewards' balancing as multiobjective optimization, where the measurement for multiple objectives is used to evaluate the agent's behavior.

B. Orientation-Preserving Pareto

To evaluate the quality of the vector-valued solution, the Pareto domination concept [37] is proposed in the multiobjective optimization domain.

Definition 1 (Pareto Domination): A solution x_u is said to dominate x_v if and only if $\forall i \in \{1, \ldots, n\}$, $f_i(x_u) \geq f_i(x_v) \wedge \exists i \in \{1, \ldots, n\}$, $f_i(x_u) > f_i(x_v)$, where $f_i\{1, \ldots, n\}$ is the ith objective function.

Based on Definition 1, a solution x_u is said to be Pareto optimal if and only if there is no solution that dominates x_u. The set of Pareto optimal solutions is called the Pareto set, and the goal of multiobjective optimization is approximating the Pareto set.

We observe that the process of multirewards' balancing is similar to the process of searching a Pareto solution in multiobjective optimization, where the main objective on this Pareto solution is better accomplished than that on any other Pareto solutions. However, previous Pareto-based algorithms are not directly applicable to rewards' balancing since they could induce many Pareto solutions that are not expected. Therefore, to converge to this Pareto solution, a new evaluator that we propose is used to evaluate how well the agent behaves. We name it as OP-Pareto [see Definition 2]. It incorporates preference information for main rewards into the original Pareto domination, aiming to guide the policy search toward more main rewards through multidirectional traction of auxiliary rewards.

Definition 2 (OP-Pareto): A full trajectory $\tau^1 = ((s_0^1, a_0^1), \ldots, (s_{k-1}^1, a_{k-1}^1))$ preference dominates another full trajectory $\tau^2 = ((s_0^2, a_0^2), \ldots, (s_{t-1}^2, a_{t-1}^2))$, denoted as $\tau^1 \succ_{pf} \tau^2$ if and only if one of the following two conditions is satisfied.

1) *τ^1 Absolute Dominates τ^2:*

$$\sum_k r_m\left(s_k^1, a_k^1\right) > \sum_t r_m\left(s_t^2, a_t^2\right).$$

2) *τ^1 Pareto Dominates τ^2:*

$$\begin{aligned} \forall i, i &\in \{m, a_1, \ldots, a_n\} \\ &\sum_k r_i\left(s_k^1, a_k^1\right) \geq \sum_t r_i\left(s_t^2, a_t^2\right) \\ \wedge\ \exists i, i &\in \{m, a_1, \ldots, a_n\} \\ &\sum_k r_i\left(s_k^1, a_k^1\right) > \sum_t r_i\left(s_t^2, a_t^2\right). \end{aligned}$$

The rationality of OP-Pareto can be explained as follows. The absolute domination in condition 1) plays a role in ensuring that the overall optimization orientation for the reward function is consistent with the goal of achieving the main objective better. Based on the concept of Pareto optimum, condition 2) encourages the agent to conduct multidirectional exploration on those other objectives when a better solution cannot be optimized on the main objective. It is reasonable that other objectives help to optimize the main objective, but it is not clear in what form they work.

According to Definition 2, if a full trajectory τ^1 is better than another full trajectory τ^2 on the main objective, then using *OP-Pareto* can still ensure that τ^1 is better than τ^2. The reverse is also true. Through inductive recursion, optimizing the reward function induced by *OP-Pareto* can guide the agent to quickly converge to the preference multiobjective region, which is exactly the purpose of the task.

More precisely, the process of rewards' balancing can be formulated in the following way: if

$$\underbrace{\left((s_0^1, a_0^1), \ldots, (s_{k-1}^1, a_{k-1}^1)\right)}_{\tau^1} \succ_{pf} \underbrace{\left((s_0^2, a_0^2), \ldots, (s_{t-1}^2, a_{t-1}^2)\right)}_{\tau^2}$$

then maximize $d_w[\tau^1 \succ_{pf} \tau^2]$ by optimizing w

$$d_w[\tau^1 \succ_{pf} \tau^2] = \sum_k r_w\left(s_t^1, a_t^1\right) - \sum_t r_w\left(s_t^2, a_t^2\right) \quad (4)$$

where $r_w(s_t, a_t) = \sum_i w_i(s_t, a_t) \cdot r_i(s_t, a_t)$, and without losing its generality, $\sum_i w_i(s_t, a_t) = 1, w_i(s_t, a_t) \in [0, 1]$. The weight vector indicates the tradeoff among the multiple objectives at each state–action pair.

C. Orientation-Preserving Reinforcement Learning

Formally, we give the following two core equations for rewards' balancing and policy optimization, respectively. Equation (5) aims to achieve rewards balance by optimizing weights so that the better the trajectory evaluated under OP-Pareto, the greater the cumulative return. Then, based on (6), the policy can be optimized by maximizing the weighted sum of reward function. In this way, the policy would be driven along the orientation that maximizes the main rewards, which is exactly the purpose of the task

$$\mathcal{J}^{w} = \sum_{\{\tau^1,\tau^2\}\subset\mathcal{T}} [I(\tau^1 \succ_{pf} \tau^2)\cdot d_w[\tau^1 \succ_{pf} \tau^2] + I(\tau^2 \succ_{pf} \tau^1)\cdot d_w[\tau^2 \succ_{pf} \tau^1]] \quad (5)$$

where $I(\tau^1 \succ_{pf} \tau^2) = \begin{cases} 1 & \text{if } \tau^1 \succ_{pf} \tau^2, \\ 0 & \text{otherwise.}\end{cases}$, and $\mathcal{T}$ denotes all possible trajectories generated by the interaction between agent and environment

$$\mathcal{J}^{\pi} = E\left[\sum_{t=0}^{\infty} \gamma^t \sum_i w_i(s_t,a_t)\cdot r_i(s_t,a_t)\right]. \quad (6)$$

However, according to (5), the key for rewards' balancing is whether $\mathcal{T}$ contains lots of diverse trajectories, especially the Pareto optimal trajectory evaluated under OP-Pareto. Intuitively, an agent can randomly interact with the environment exhaustively until finding enough trajectory for rewards' balancing. However, this process would be extremely time-consuming especially in large-scale environments. Instead, we propose an iterative optimization framework called OP-RL, which sequentially performs rewards' balancing [see (5)] and Policy Optimization [see (6)] (i.e., in each iteration, the policy π_t interacts with the environment to produce a set of different trajectories $\mathcal{T}_t$, which are used to balance rewards, and then, the policy π_t is updated by optimizing the balanced rewards). Through iteratively optimizing (5) and Eqn (6), OP-RL aims to gradually infer the correct relationship between rewards. In the following, we will replace $\mathcal{T}$ with $\mathcal{T}_t$ to represent the trajectories in the tth iteration.

D. Convergence and Time Complexity Analysis

Although Section IV-C introduced the iterative optimization framework for rewards' balancing, the reward function learned in (5) would change in an unorganized way if the quality of trajectories sampled in each iteration cannot be controlled. As a result, the balanced rewards would be constantly changing. Therefore, it is necessary to analyze the convergence of OP-RL. We first model the convergence analysis problem as finding Pareto optimal trajectories evaluated under OP-Pareto. Then, through abstracting $\mathcal{T}_t$ as a Markov chain, two sufficient conditions for convergence are obtained. Furthermore, we analyze the boundary of OP-RL's time complexity in theory.

Lemma 1: The convergence of OP-RL lies in that optimal trajectories evaluated under OP-Pareto are stably obtained as the iteration proceeds.

The goal of rewards' balancing is to optimize a reward function that makes the cumulative return of optimal trajectories greater than that of any other suboptimal trajectories. According to (5), once optimal trajectories are obtained, rewards would be balanced toward the goal by comparing with other suboptimal trajectories. On the other hand, optimal trajectories of the task are optimal trajectories evaluated under OP-Pareto. Therefore, we model the convergence analysis problem as to whether optimal trajectories under OP-Pareto are stably obtained as the iteration proceeds.

Definition 3 (Random Process in T_t): Let the iteration process of trajectories be a random process $\{T_t\}_{t=1}^{+\infty}$, where $T_t = \bigcup_{i=1}^{M}\{\tau_{t,i}\}$, and $\tau_{t,i}$ is the ith sampled trajectory in the tth iteration.

Lemma 2: $\{T_t\}_{t=1}^{+\infty}$ is a Markov chain.

T_t is sampled through the policy π_t with some random exploration, and the rewards used to optimize π_t are balanced only by T_{t-1}. Therefore, we have $P(T_t|T_{t-1},\ldots,T_1) = P(T_t|T_{t-1})$.

Definition 4 (Optimal State Space): Let Ω_T be the state space of $\{T_t\}_{t=1}^{+\infty}$. $\Omega_T^{\text{opt}} \subset \Omega_T$ is called the optimal state space if $\forall T \in \Omega_T^{\text{opt}}, \exists\tau^* \in T$, where τ^* is an optimal trajectory evaluated under OP-Pareto.

According to Definition 4, if $P(T_t \in \Omega_T^{\text{opt}}) = 1$, it indicates that optimal trajectories are obtained with probability 1 in the tth iteration. Similarly, $\lim_{t\to+\infty} P(T_t \in \Omega_T^{\text{opt}}) = 1$ indicates that optimal trajectories are obtained with probability 1 after infinite iterations.

Theorem 1 (Convergence Conditions): If $P(T_{t+1} \notin \Omega_T^{\text{opt}}|T_t \in \Omega_T^{\text{opt}}) = 0$ and $P(T_{t+1} \in \Omega_T^{\text{opt}}|T_t \notin \Omega_T^{\text{opt}}) > 0$, where $t = 1, 2, \ldots$, then $\lim_{t\to+\infty} P(T_t \in \Omega_T^{\text{opt}}) = 1$.

Theorem 1 gives two sufficient conditions for T_t to reach Ω_T^{opt} with probability 1. The proof of the Theorem is in Appendix A. A detailed explanation is given as follows.

1) $P(T_{t+1} \notin \Omega_T^{\text{opt}}|T_t \in \Omega_T^{\text{opt}}) = 0$ indicates that, once the random process T_t reaches the optimal state space Ω_T^{opt}, it will be adsorbed in Ω_T^{opt} and will never come out. To meet this condition, OP-RL should always keep the optimal trajectories contained in T_t into T_{t+1} in each iteration.
2) $P(T_{t+1} \in \Omega_T^{\text{opt}}|T_t \notin \Omega_T^{\text{opt}}) > 0$ indicates that the random process T_t has a nonzero probability to reach the optimal state space Ω_T^{opt} at any time t. In particular, in the $(t+1)$th iteration, the policy π_{t+1} can sample optimal trajectories with nonzero probability and store them into the T_{t+1}. To meet this condition, OP-RL should be supplemented by some random exploration when sampling with $\pi^{t+1}(a|s)$ so that, $\forall s \in S, a \in A, \pi^{t+1}(a|s) > 0$. Here, we define the optimal trajectory as $\tau^* = \{s_1^*, a_1^*, s_2^*, a_2^*, \ldots, s_h^*, a_h^*\}$; then, $P(T_{t+1} \in \Omega_T^{\text{opt}}|T_t \notin \Omega_T^{\text{opt}}) = \prod_{j=1}^{h} \pi^{t+1}(a_j^*|s_j^*)P(s_{j+1}^*|s_j^*, a_j^*) > 0$. Thus, if, $\forall s \in S, a \in A, \pi^{t+1}(a|s) > 0$, optimal trajectories can be sampled into $\{T_t\}_{t=1}^{+\infty}$ with nonzero probability in each iteration.

The previous result gives sufficient conditions for the convergence of OP-RL. In practice, it is more expected that how many iterations OP-RL can find optimal trajectories. Next, we will give a theoretical analysis of the time complexity of our algorithm.

Theorem 2 (Time Complexity Analysis): Let ϵ-greedy exploration be used in OP-RL. $|A|$ denotes the size of action space. $E[\mu]$ denotes the expected convergence time of the random process $\{T_t\}_{t=1}^{+\infty}$ reaching the optimal state space Ω_T^{opt}. $E[\mu]$ can be bounded by

$$(1-\lambda_1)\left\{P(\tau^*)\left[1-\epsilon+\frac{\epsilon}{|A|}\right]^h\right\}^{-1} \leq E[\mu] \leq (1-\lambda_1)\left\{P(\tau^*)\left[\frac{\epsilon}{|A|}\right]^h\right\}^{-1} \quad (7)$$

where $\lambda_1 = P(T_1 \in \Omega_T^{\text{opt}})$ and $P(\tau^*) = \prod_{j=1}^{h} P(s_{j+1}^*|s_j^*, a_j^*)$.

According to Theorem 2, the expected time for optimal trajectories entering into the buffer is related to the complexity of the task. The larger the action space and the longer the time step of the episode, the longer the expected convergence time. The proof of the theorem is in Appendix B.

V. Algorithm for Orientation-Preserving Reinforcement Learning

In this chapter, we propose a complete algorithm for OP-RL. From the above theoretical analysis, we can see that two sufficient conditions for algorithm convergence are: 1) $P(T_{t+1} \notin \Omega_T^{\text{opt}}|T_t \in \Omega_T^{\text{opt}}) = 0$ and 2) $P(T_{t+1} \in \Omega_T^{\text{opt}}|T_t \notin \Omega_T^{\text{opt}}) > 0$. Next, we would design the algorithm based on the theoretical results.

At each point, our method maintains a policy $\pi : s \to a$ and a weight estimation $W : s \times a \to \vec{w}$, each parameterized by a deep neural network. Through evaluating the sampled trajectories using *OP-Pareto* [see Definition 2], the weight estimation W can be optimized via supervised learning to fit the evaluation, so as to ensure that the better the agent's behavior, the greater the cumulative return. Then, a scalarization reward function can be acquired by weighting the sum of the multiple rewards. In this way, the policy is updated to optimize the learned reward function. We can train this policy using any RL algorithm that is appropriate for the domain. The algorithm framework is shown in Fig. 2.

In particular, the two networks are updated iteratively by three processes, as described in Algorithm 1.

Step 1: The policy π interacts with the environment with some random exploration to produce a set of different full trajectories $\{\tau^1, \ldots, \tau^m\}$. The randomness makes it possible to find any trajectory and so do optimal trajectories. Then, put these sampled trajectories into the trajectory buffer $\mathcal{D}_{\text{trj}}$.

Step 2: Randomly select pairs of trajectories (τ^i, τ^j) from the buffer $\mathcal{D}_{\text{trj}}$, and compare them using OP-Pareto; then, the parameters of the mapping W are optimized via supervised learning to fit the comparisons.

Step 3: The parameters of π are updated by a traditional reinforcement learning algorithm in order to maximize the discount sum of learned rewards $r(s_t, a_t) = \sum_i w_i(s_t, a_t) \cdot r_i(s_t, a_t)$.

However, due to the randomness of sampling in Step 1, the quality of trajectories sampled in each iteration cannot be controlled. As a result, the reward function learned in Step 2 would change in an unorganized way as the iteration progresses, which makes the algorithm unable to converge. To this end, we design a *elitist retention buffer* to overcome the influence of randomness. We show that this meets the convergence condition 1 (i.e., $P(T_{t+1} \notin \Omega_M^{\text{opt}}|T_t \in \Omega_M^{\text{opt}}) = 0$).

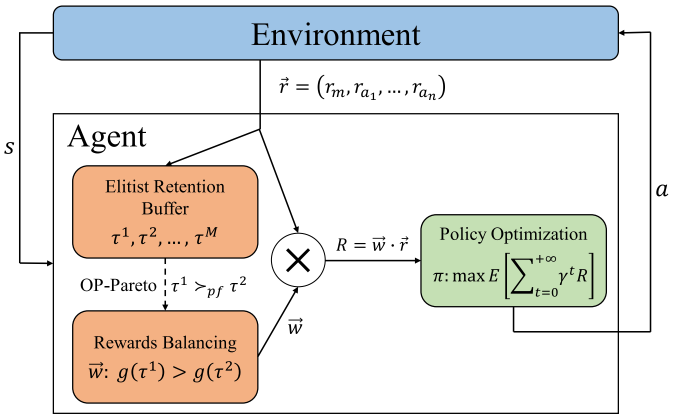

Environment
$\vec{r} = (r_m, r_{a_1}, \ldots, r_{a_n})$
Agent
s
a
Elitist Retention Buffer
$\tau^1, \tau^2, \ldots, \tau^M$
OP-Pareto
$\tau^1 >_{pf} \tau^2$
Rewards Balancing
$\vec{w}$: $g(\tau^1) > g(\tau^2)$
$\vec{w}$
$R = \vec{w} \cdot \vec{r}$
Policy Optimization
π: max $E\left[\sum_{t=0}^{+\infty} \gamma^t R\right]$

Fig. 2. Algorithm framework for automatic rewards' balancing. In each turn, the policy π interacts with the environment to sample some trajectories that are then evaluated by OP-Pareto. Then, weights are optimized to fit the evaluation for rewards' balancing. The policy π is updated to maximize the balanced rewards.

Algorithm 1 OP-RL

```
Input:
1: Main objective reward: $r_m$
2: $n$ other objective reward: $r_{a_1}, \ldots, r_{a_n}$
3: Trajectory buffer $D_{trj}$, Capacity $M$, Per Retaining $K$
Output:
4: Policy network $\pi$, Weight estimation network $W$
5:
6: for $i = 0 \to Iteration - 1$ do
7:     Retain top $K$ optimal trajectories in $D_{trj}$.
8:     Sample $M - K$ new trajectories using $\pi$ and put them
       into $D_{trj}$.
9:     for $j = 0 \to WeightUpdate - 1$ do
10:        Sample a pair of trajectories from $D_{trj}$.
11:        Optimize $W$ using Eqn (9).
12:    end for
13:    for $q = 0 \to PolicyUpdate - 1$ do
14:        Compute $r = \sum_i w_i \cdot r_i$ using $W$.
15:        Update $\pi$ using $r$ based on Eqn (6).
16:    end for
17: end for
```

A. Elitist Retention Buffer

If only the sampled trajectories in the current iteration are kept in the buffer, there is no guarantee that the reward function would be gradually improved due to the randomness of sampling. An effective way is to make use of past sampled trajectories so that good trajectories evaluated under OP-Pareto are always kept in the buffer. Directly, we can make the replay buffer larger so that early trajectories sampled through random exploration are still saved. However, it seems impractical for

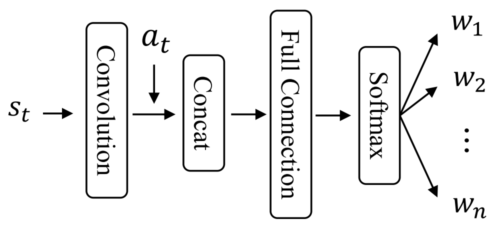


Fig. 3. Network architecture of weight estimation.

two reasons: 1) unless the buffer is infinite, older trajectories will eventually be erased before acquiring the true reward function and 2) even if all past trajectories are still present, learning from an increasing number of trajectories remains hard because the learning speed of each iteration will be slower and slower. Therefore, instead of retaining all past trajectories, the following two steps are taken to ensure effective learning of weight vector: 1) retaining top K preference optimal trajectories in the current buffer and 2) replacing the remaining $M-K$ trajectories in the current buffer with new trajectories sampled by the current policy π. The purpose of the first step is to improve the utilization rate of preference trajectories and push weight learning toward the preferred direction. The second step can enable the reward function to be effectively learned in the case of the limited buffer.

On the other hand, if there is a greater diversity of trajectories, it is more helpful for weight learning. OP-RL adopts ϵ-greedy exploration in Step 1. This meets the convergence condition 2 (i.e., $P(T_{t+1} \in \Omega_M^{\text{opt}} | T_t \notin \Omega_M^{\text{opt}}) > 0$).

B. Weight Estimation Network

We can interpret the weight estimation W as a preference predictor if we view w as a latent factor explaining which objective the agent should prefer at each time. The network architecture of the weight estimation is shown in Fig. 3. Instead of optimizing w based on (5), we assume that the probability of preferring a trajectory τ^i depends exponentially on the value of the latent reward summed over the length of the trajectory. In this way, (8) is used to replace (5), which has been found to work well in the work [26]

$$P[\tau^1 \succ_{pf} \tau^2] = \frac{exp \sum r(s_t^1, a_t^1)}{\exp \sum r(s_t^1, a_t^1) + exp \sum r(s_t^2, a_t^2)} \quad (8)$$

where $r(s_t, a_t) = \sum_i w_i(s_t, a_t) \cdot r_i(s_t, a_t)$.

We optimize w to minimize the cross-entropy loss

$$\mathcal{J}^w = - \sum_{\{\tau^1, \tau^2\} \subset \mathcal{D}_{\text{trj}}} [I(e^1 \succ_{pf} e^2) \cdot \log P[\tau^1 \succ_{pf} \tau^2] + I(e^2 \succ_{pf} e^1) \cdot \log P[\tau^2 \succ_{pf} \tau^1]]. \quad (9)$$

Although (8) and qn (9) are also adopted in preference learning proposed by Christiano *et al.* [26], the main difference lies in the comparison between trajectories. In the work [26], a human has to provide feedback on our agent's behavior during the whole learning process. It consumes a lot of labor costs and is not suitable for solving complex tasks. In contrast, OP-Pareto that we proposed replaces the role of experts in

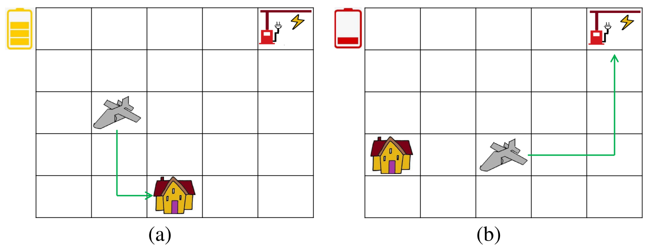


Fig. 4. Sketch of (a) charging and (b) delivery. The purpose of the agent is to control the UAV to deliver as many packages as possible by weighing the importance of delivery and charging.

evaluating how well the agent behaviors. This can significantly improve the efficiency of solving the optimization problem.

VI. Experiments and Results

In the following experiments, we aim to answer the following questions.

1) Whether it will lead to difficulty in learning a good policy if the main rewards are considered only?
2) Whether OP-RL can learn the correct time-variant relationship among objectives in a self-supervised manner?
3) To what extent we can improve performance on accomplishing tasks by learning to balance rewards, compared with those RLs with heuristically designed rewards (i.e., fixed weights and dynamic weights)?

We first answer these questions on a simple grid world environment involving weighing the relative importance of two objectives. Then, we empirically evaluate our approach on Doom, a first-person-shooting (FPS) game in a 3-D environment, where the difficulty of rewards' balancing is exacerbated due to the high-dimensional input and more objectives.

A. Environments

1) Charging and Delivery: To identify the quality of our algorithm, we design a new benchmark problem: charging and delivery. A sketch of the game is shown in Fig. 4. The agent has four actions: moving up, down, right, and left. The purpose is to control an unmanned aerial vehicle (UAV) to deliver as many packages as possible in a 5 * 5 grid world. There is only one delivery location at a time, and the next delivery location appears on the other grid after the UAV reaches the current delivery location. However, the UAV cannot fly for too long due to its limited battery capacity (i.e., each move consumes one unit of electricity). Therefore, a charging station is set on the upper right of the grid world. The agent can get 20 units of electricity per charge but at the cost of consuming five time steps. In this case, the agent needs to weigh the importance of delivery and charging at every moment to deliver as many packages as possible within 100 time steps.

2) Doom Game: Kempka *et al.* [9] employ a first-person perspective in a semirealistic 3-D world. As shown in Fig. 5, we use the modification of "defend the line" [9] as our experimental environment. In this scenario, the agent is spawned along the long corridor. Three shooting monsters are spawned

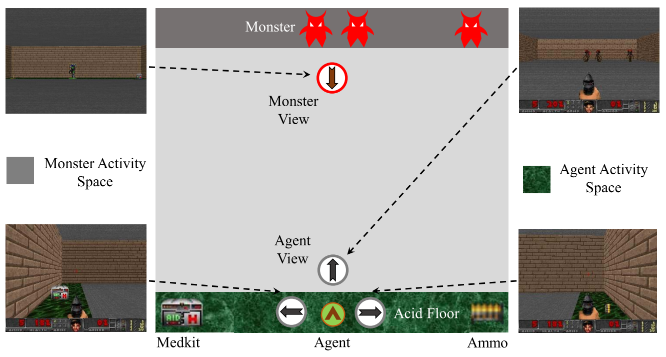


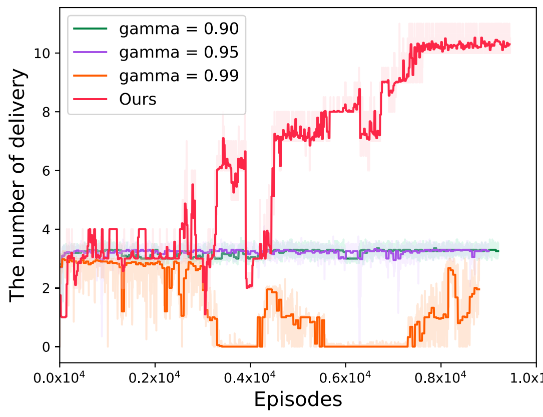
Fig. 5. Screenshots of modified "defend the line" in Doom. The purpose of the agent is to fighting as many monsters as possible by weighing the importance of fighting, collecting clips, and medkits.

randomly somewhere along the opposite corridor and try to kill the agent with fireballs. Monsters can also be killed by the agent after a single shot. After dying, each monster is respawned after a while. The agent can only move along his corridor, and medkits and clips are regularly generated at each end of the corridor. Clips replenish the agent's bullets, helping to kill more monsters. Medkits heal some portions of the agent's health, helping to survive. The purpose of the agent is to maximize the number of monsters killed. Thus, the agent must weigh the importance of the three objectives all the time: fighting monsters, collecting medkits, and collecting clips. In addition, to make the task more interesting and more challenging, the floor of the corridor on the agent's side is covered with acid, which hurts the agent periodically. This further increases the survival crisis of the agent.

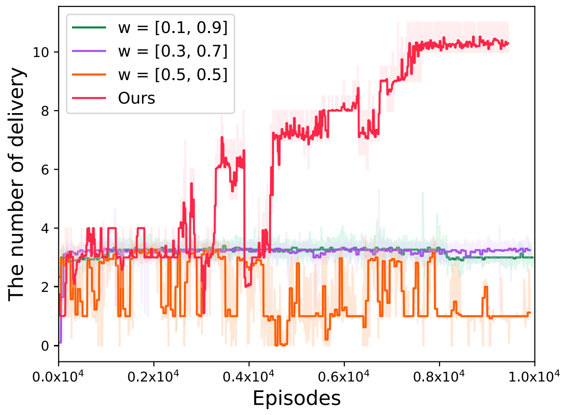
In particular, the agent can perform one of the four standard actions: moving forward, turning left, turning right, and attack. The four actions can be combined, which yields seven combination actions: moving forward + turning left, moving forward + turning right, moving forward + atack, turning left + attack, turning right + attack, moving forward + turning left + attack, and moving forward + turning right + attack. In this experiment, we used four standard actions and seven combination actions as the action set. In the beginning, the agent has five ammo and 20 healths. Each time a clip is picked up, the agent replenishes five ammo. The agent has a maximum of five ammo at a time. Each time a medkit is picked up, the agent replenishes 15 healths. The agent can have up to 40 healths at a time. Time steps per episode are limited to 1000 frames.

B. Results in Charging and Delivery

1) Experimental Settings: There are two objectives to consider: delivery and charging. We design a reward function for each objective as follows: giving a $r_c = +1$ reward for one charging, giving a $r_d = +1$ reward for one delivery, and 0 otherwise. In this game, deep Q-learning [1] is used to optimize the policy. More details of this experiment are described in Appendix C, including model architectures and environment configuration.



(a)



(b)

Fig. 6. Performance comparison on the charging and delivery game. (a) We compare our algorithm with those that optimize the main rewards only. (b) We further compare our algorithm with those that optimize the reward function $r = w_1 * r_1 + w_2 * r_2$ with different fixed weights.

2) Compared With the Main Rewards: We first compare our algorithm with those that optimize the main rewards only, and the latter is set with different discount factors. The discount factor is set to 0.95 in our algorithm. The learning curves of all algorithms are shown in Fig. 6(a). The vertical axis represents the completion of the main objective. We can see that, although training with learned reward functions tends to be less stable and has a higher variance, our algorithm gives a marginal improvement over those that consider only the main objective. Specifically, our algorithm can complete the delivery of ten packages, while others can only complete three delivery. The reason for the poor performance of optimizing the main rewards is that the reward has design flaws, as explained in Section III-B.

In addition, the discount factor has a crucial effect on the learning processes when only the main rewards are considered. When we use a discount factor of 0.90 or 0.95, the algorithm quickly converges to a highly suboptimal solution that is extremely myopic. The reasons are given as follows. Although charging helps more delivery, going to charge undoubtedly takes more time steps without any reward. Thus, even if the agent's trajectory has more delivery, its cumulative discounted return is less when a low discount factor is used. It is clear

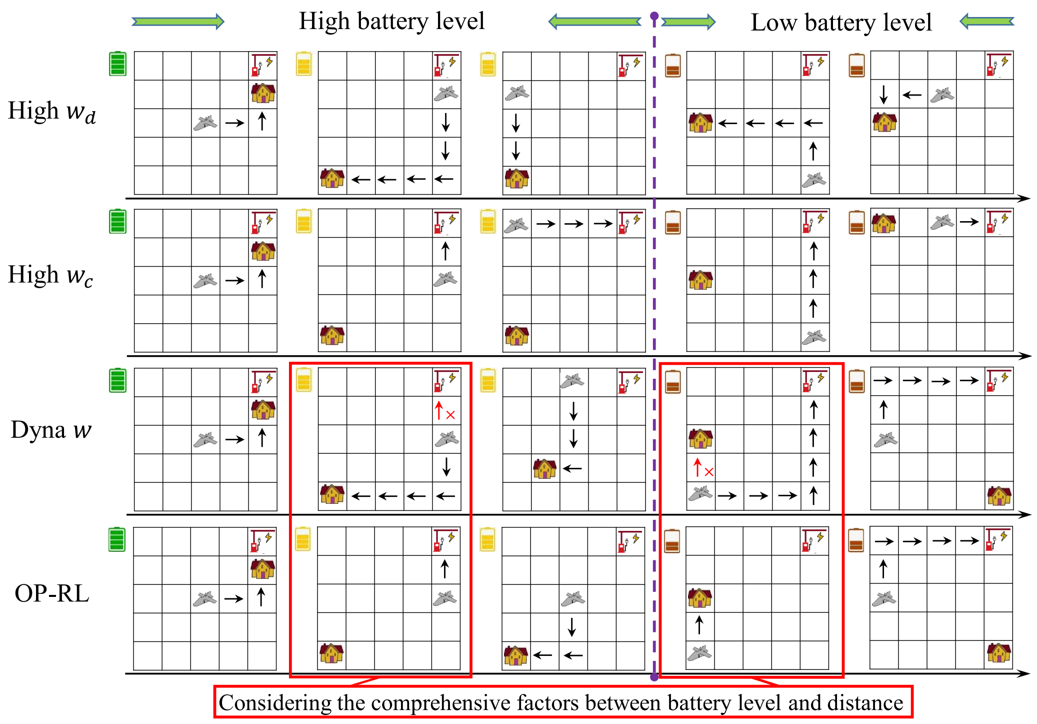

High battery level
Low battery level
High w_d
High w_c
Dyna w
OP-RL
Considering the comprehensive factors between battery level and distance

Fig. 7. Schematic of the policies under four weight settings. Each figure represents the decisions that the agent will make under different power levels. **Top Row:** when w_d is high, the agent is overly inclined to deliver and dies until running out of power. **Second Row:** when w_c is high, the agent tends to charge all the time until the episode is finished. **Third Row:** the agent with hand-designing dynamic weights makes a decision only based on the battery level. **Bottom Row:** OP-RL considers the comprehensive factors between battery level and distance.

that a larger discount factor is desired. However, it causes difficulties in algorithm convergence. As the value is set to 0.99, even though the short-sightedness is relieved at this time, the algorithm is too hard to converge due to the huge training variance.

3) Compared With Fixed Weight Settings: We compare OP-RL with those that optimize the reward function $r = w_c * r_c + w_d * r_d$ with different fixed weights in Fig. 6(b). The discount factor is set to 0.95 for all experiments. We can see that our algorithm for weight learning gives a consistent improvement over those that use fixed weight. When the weight for charging is low, such as 0.1 or 0.3, the algorithm with fixed weight performs almost the same as those only consider the main rewards. The reason may be that the weight is too small to allow the agent to pay attention to charging. When the weight for charging is high, such as 0.5, the algorithm performs worse on accomplishing the main objective. The reason is that the agent cannot distinguish which objective is more important.

Fig. 7 shows how the learned policies accomplish each objective along the trajectory. We can see that the policies with a high weight for delivery always focus on delivery only and ignore charging. On the contrary, the policy is too biased toward charging when it has a high weight for charging. Therefore, it is hard for the agent to weigh the importance of each objective overtime when the fixed weight is used.

4) Compared With Dynamic Weight Settings: We further compare OP-RL with those RLs with hand-designing dynamic weights. According to our knowledge of the game, the following reward function is designed:

$$\begin{cases} w_d = 1, & \text{if battery level} \geq \delta, \text{else} w_d = -1 \\ w_c = -1, & \text{if battery level} \geq \delta, \text{else} w_c = 1. \end{cases} \quad (10)$$

The dynamic rewards encourage the agent to deliver at a high battery level and charge at a low battery level. The threshold δ is used to distinguish between high and low battery levels. To compare the impact of different thresholds on policy learning, we set up multiple sets of dynamic rewards. Table I shows the policy's performance on each objective under different δ's. As shown in Table I, δ has a great influence on the policy. Although the policy with $\delta = 9$ performs better on the main objective than those with other δ, its performance

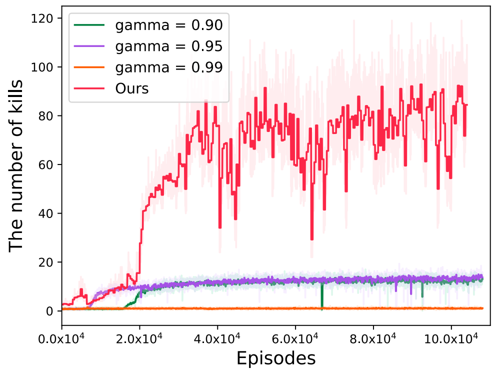
TABLE I
PERFORMANCE ON EACH OBJECTIVE WITH DYNAMIC WEIGHT SETTINGS IN CHARGING AND DELIVERY

Hyper-parameter	**Number of delivery**	Number of charging
$\delta = 5$	3	0
$\delta = 7$	6	4
$\delta = 9$	8	5
$\delta = 11$	7	5
$\delta = 13$	1	7
$\delta = 15$	1	9
OP-RL	**11**	5

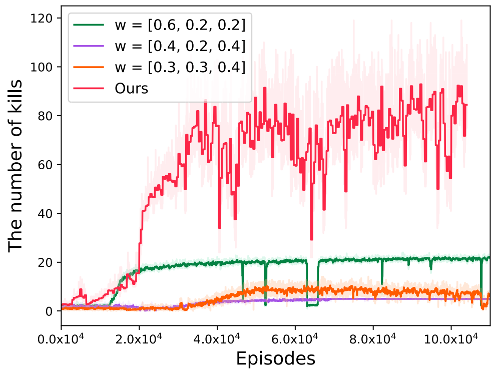
TABLE II

VALUE OF LEARNED WEIGHT ALONG THE TRAJECTORY IN OP-RL. THE NOTATION i/j IN THE ROW OF "TASK PROCESS" INDICATES THE COMPLETION OF THE iTH DELIVERY AND THE jTH CHARGING

Battery Level	16	15	13	11	5	15	9	7	16	9	5	15	11	8	14	11
Task Process	1/0	1/1	2/1	3/1	3/2	4/2	5/2	5/3	6/3	7/3	7/4	8/4	9/4	9/5	10/5	11/5
Charging w_c	0.31	**0.78**	0.08	0.20	**0.98**	0.00	0.38	**0.94**	0.00	0.05	**0.92**	0.00	0.04	**0.86**	0.02	0.00
Delivery w_d	**0.69**	0.22	**0.92**	**0.80**	0.02	**1.00**	**0.62**	0.06	**1.00**	**0.95**	0.08	**1.00**	**0.96**	0.14	**0.98**	**1.00**



(a)



(b)

Fig. 8. Performance comparison on the Doom game. (a) We compare our algorithm with those that optimize the main rewards only. (b) We further compare our algorithm with those that optimize the reward function $r = w_{\text{km}} * r_{\text{km}} + w_{\text{cc}} * r_{\text{cc}} + w_{\text{cm}} * r_{\text{cm}}$ with different fixed weights.

is still not as good as OP-RL. The reason is that the above dynamic rewards only consider the impact of battery level on decision-making, while other factors are not taken into consideration, such as "distance." As shown in the red box in Fig. 7, the agent with hand-designing dynamic weights makes a decision only based on the battery level, even if the delivery location (or the charging station) is near to the agent. This leads to the destruction of the optimal policy in the time-limited task. However, it is often hard for experts to consider all factors when designing the reward, especially in complex tasks. On the other hand, even if all influencing factors are considered, the complicated logic among these factors may be too vague to clarify.

Compared with hand-designing dynamic weights, OP-RL finds an almost optimal policy to accomplish the main objective through learning to reward balancing in a self-supervised manner. Table II shows the value of learned weight along the trajectory in our algorithm. We observe that w_d is much larger than w_c when the power is sufficient. This means that the agent should go to deliver at that time. When the power is insufficient, w_c becomes the main weight, which means that charging is imminent. This result implies that the learned weight indeed captures the relationship between delivery and charging.

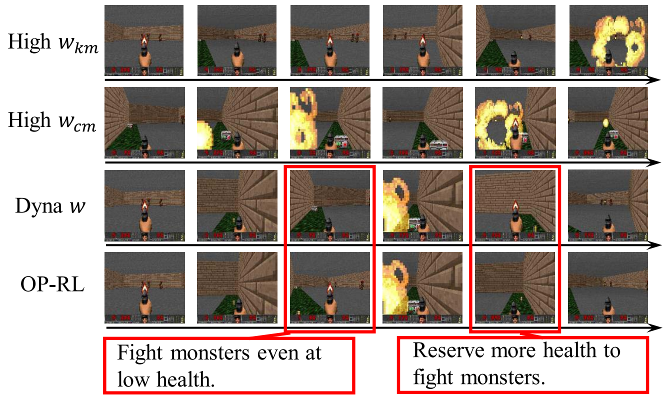

High w_{km}
High w_{cm}
Dyna w
OP-RL
Fight monsters even at low health.
Reserve more health to fight monsters.

Fig. 9. Illustration of how the agent makes decision at different stage of the game. **Top Row:** agents tend to fight monsters while collecting clips. **Second Row:** agent tends to collect medkits all the time. **Third Row:** agent makes decision only based on health level. **Bottom Row:** OP-RL considers the comprehensive factors between health level and monster attack level.

C. Results in Doom

1) Experimental Settings: Three objectives need to be considered in this game. We design a reward function for each objective as follows: giving a $r_{\text{km}} = +1$ reward for killing a monster, giving a $r_{\text{cc}} = +1$ reward for collecting a clip, and giving a $r_{\text{cm}} = +1$ reward for collecting a medkit. We can use any RL algorithm that is appropriate for the domain. Considering that the learned reward function may be nonstationary, we prefer RL algorithms that are robust to changes in the reward function. In this experiment, we use advantage actor–critic (A2C) [38] to optimize the policy. More details of this experiment are described in Appendix C.

2) Compared With the Main Rewards: In this game, the main objective is killing monsters, and the two other objectives are collecting medkits and clips, respectively. First, we compare OP-RL with baselines that optimize the reward function only from the main objective. The baselines are set with different discount factors. The discount factor is set to 0.95 in OP-RL. The learning curves of all algorithms are shown in Fig. 8(a). As expected, OP-RL gives a marginal improvement over those that consider only the main rewards.

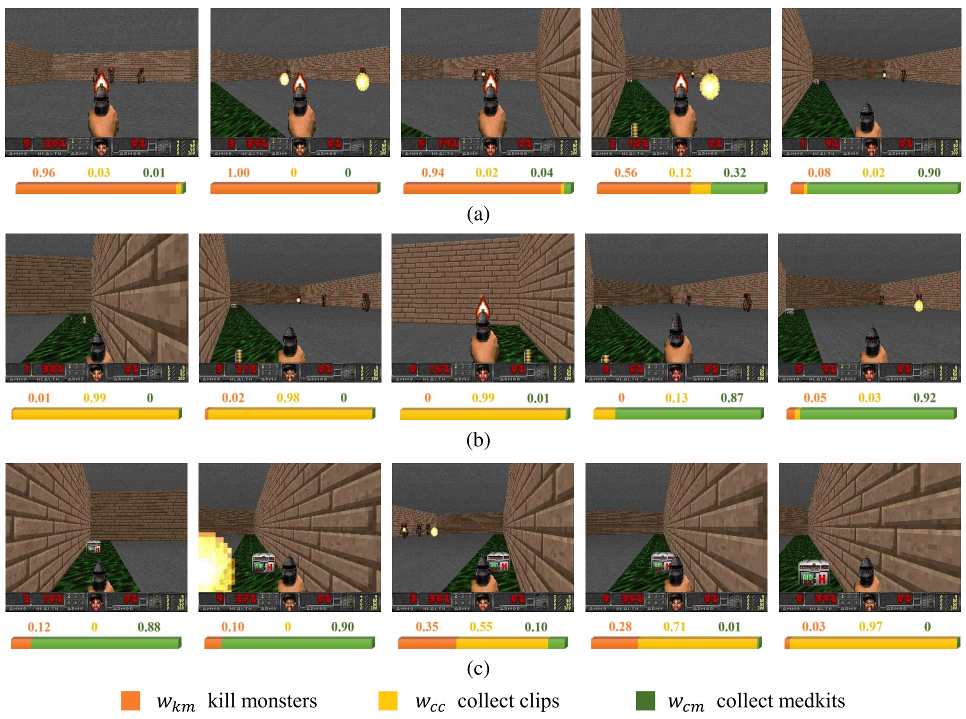


Fig. 10. Illustration of how the weights learned by OP-RL discriminate at different stages of the game. (a) Visualization of the weight when killing monsters in different stages. (b) Visualization of the weight when collecting clips in different stages. (c) Visualization of the weight when collecting medkits in different stages.

Specifically, OP-RL can kill an average of 80 monsters, and policies optimized by baselines can only kill up to 15 monsters. From this perspective, our algorithm is almost six times better than others. This is a powerful verification that it is almost impossible to drive a good policy through only the traction of the main rewards.

In addition, we also empirically evaluate the impact of discount factors on the optimization process when considering the main rewards alone. When we use a discount factor of 0.90 or 0.95, the algorithm quickly converges to a highly suboptimal solution. The discount factor of 0.9 leads to faster convergence. The reasons are given as follows. As stated in Section III-B, one of the reasons is that a lower discount factor can lead to a myopic policy. Another major reason is the problem of temporal credit assignment. Due to the huge state–action space, it is difficult to deduce other objectives and their importance relative to the main objective only by returning the reward signal from the main objective. In other words, the reward is extremely sparse, which makes it very difficult for the agent to learn favorable actions. Thus, even with a larger discount factor, it still cannot alleviate the difficulty of temporal credit assignment. Moreover, it is frequently observed that larger discount factors achieve worse results [35]. As shown in Fig. 8(a), when the value is set to 0.99, the performance cannot be improved for an extremely long time.

3) Compared With Fixed Weight Setting: We compare OP-RL with those that optimize the reward function $r = w_{\mathrm{km}} * r_{\mathrm{km}} + w_{\mathrm{cc}} * r_{\mathrm{cc}} + w_{\mathrm{cm}} * r_{\mathrm{km}}$ with different fixed weights in Fig. 8(b). The discount factor is set to 0.95 for all experiments. We can see that OP-RL gives significant improvement over those that use fixed weight in this more complex environment. According to our knowledge of the game, the most important objective is to fight monsters, and collecting medkits is just as important as collecting clips. Thus, referring to the reward setting in the work [39], we set the weight vector as $w_{\mathrm{km}} = 0.6, w_{\mathrm{cc}} = w_{\mathrm{cm}} = 0.2$. However, experimental results show that this setting performs only slightly better than those that only consider the main rewards. As shown in the first row in Fig. 9, the agent tends to fight monsters all the time rather than collect medkits. Therefore, the agent will eventually die because of losing all health. In order to encourage the agent to collect medkits, we increase the weight for this objective, and the weight vector is $\vec{w} = [0.4, 0.2, 0.4]$. Unexpectedly, experimental results show that this setting even performs worse. The second row in Fig. 9 shows visualization policies. At this time, the agent tends to collect medkits rather than fight.

4) Compared With Dynamic Weight Setting: We further compare OP-RL with those RLs with hand-designing dynamic weights. Referring to the reward designing in the work [10], [39], the following two classes of dynamic reward functions are designed:

$$w_{\mathrm{km}} = w_{\mathrm{cc}} = -w_{\mathrm{cm}} = \begin{cases} 1, & \text{if} h > \delta \\ -1, & \text{otherwise} \end{cases} \tag{11}$$

$$\begin{cases} w_{\mathrm{km}} = M, \quad w_{\mathrm{cc}} = \dfrac{2}{c+1}, w_{\mathrm{cm}} = -1, & \text{if} h > \delta \\ w_{\mathrm{km}} = -M, \quad w_{\mathrm{cc}} = -1, w_{\mathrm{cm}} = \dfrac{10}{h+1}, & \text{otherwise} \end{cases} \tag{12}$$

TABLE III
PERFORMANCE ON EACH OBJECTIVE WITH DYNAMIC WEIGHT SETTINGS IN DOOM

		The main objective	The other objectives	
	Hyper-parameter	**Num of kills**	Num of clips	Num of medkits
Dyna(11)	$\delta = 5$	36	17	4
	$\delta = 8$	24	13	4
	$\delta = 10$	60	32	8
	$\delta = 12$	28	15	9
	$\delta = 20$	6	5	10
Dyna(12)	$M = 1, \delta = 10$	22	25	9
	$M = 3, \delta = 10$	50	23	7
	$M = 5, \delta = 10$	52	20	10
	Human players	67	34	20
	OP-RL	**84**	29	10

where c and h, respectively, denotes the agent's ammunition and health level. Both of the two rewards encourage the agent to fight monsters, collect clips at high health, and collect medkits as low health. The threshold δ is used to distinguish high and low health levels. Furthermore, (12) regards w_{cc} and w_{cm} as functions of c and h, respectively, which makes the policy more sensitive to collect clips and medkits. Table III shows the policy's performance on each objective under different hyperparameters (δ and M). In comparison, OP-RL still performs best among all RLs. The reason is that the above dynamic rewards only consider the impact of health level on decision-making, while other factors are not taken into consideration, such as "monster attack level." As shown in the left red box in Fig. 9, OP-RL can fight monsters even at low health if the monster attack level is low. Moreover, OP-RL learned a smarter policy, such as reserving more health to fight monsters. This makes it possible for the agent to kill more monsters after each health supply. In a word, it is often hard for experts to consider all factors when designing the reward [40], [41]. On the other hand, OP-RL also outperforms human players, and its performance on each objective is appropriate. This is a piece of very strong evidence that the agent understands the purpose of the task.

In order to show how the weights discriminate at different stages of the game, we visualize the results in Fig. 10. From Fig. 10(a) and (b), we can see that the weights of fighting monsters w_{km} and collecting clips w_{cc} both are positively related to the agent's health. The value for weight $w_{\text{km}}, w_{\text{cc}}$ is extremely high when the agent's health is high. However, these weights are low if the health is low. This is almost the same as our perception of games. On the other hand, Fig. 10(c) shows that the weight of collecting medkits is large in most cases unless the agent keeps collecting medkits without any fighting. Considering that the agent is in an extremely dangerous environment, this is a very intelligent policy. We observed that this is almost the same as the policy of human players. Therefore, this is a powerful verification that demonstrates the advantages of OP-RL in balancing main and auxiliary rewards.

VII. CONCLUSION AND DISCUSSION

A. Significance and Novelty

Many difficulties exist in RL algorithms, such as low sample efficiency and long convergence time, among others. Adding auxiliary rewards is an efficient way to tackle the difficulties. However, the optimality of policy learning relies on the balancing between main and auxiliary rewards, which still remain a key challenge. Although rewards' balancing has been studied for many years, existing work cannot provide a principled method to avoid the interference of auxiliary rewards on pursuing the main rewards, which leads to the destruction of the optimal policy. In this article, we explicitly formulate the problem of rewards' balancing as searching for a Pareto optimal solution, with the overall objective of preserving the policy's search orientation for the main rewards. Furthermore, we establish an iterative learning framework for rewards' balancing and theoretically analyze its convergence and time complexity. Our theory and algorithm make a step toward optimality-preserving rewards' balancing in RLs.

B. Open Questions and Future Work

Although our work develops a method for automatic rewards' balancing with theoretical guarantees, the following questions remain unanswered.

Can OP-RL be applied to tasks with large state and action spaces? The convergence of OP-RL lies in that optimal trajectories evaluated under OP-Pareto are stably obtained as the iteration proceeds. According to Theorem 2, the expected convergence time of OP-RL is related to the length of the episode, action space, and exploration strategy. Thus, to apply OP-RL to tasks with large state space and action space, the following three methods may be feasible ways to improve the convergence speed: 1) perform compression in time with temporally abstract actions [42], [43] so that the agent can make decisions at a large time scale instead of at each time step; 2) reduce the size of the action space based on the action compression method [44], so as to remove meaningless actions in each decision; and 3) improve the exploration strategy, so as to increase the probability of finding the optimal trajectory in each iteration. However, this is still an open issue in RL.

Can OP-RL be applied to RL tasks that do not provide auxiliary rewards in advance? In this article, we consider the task setting where each auxiliary reward is provided; only their relationship with the main rewards is unknown. However, when faced with complex tasks, it is hard for experts to identify those objectives that are beneficial to achieve the main objective and assign corresponding rewards to them. In fact, our work can be extended to those tasks where RLs can produce auxiliary rewards automatically [7], [22].

APPENDIX A
PROOF OF THEOREM 1

Since $P(T_t \in \Omega_T^{\text{opt}} | T_{t-1} \notin \Omega_T^{\text{opt}}) > 0$, $P(T_t \notin \Omega_T^{\text{opt}} | T_{t-1} \notin \Omega_T^{\text{opt}}) = 1 - P(T_t \in \Omega_T^{\text{opt}} | T_{t-1} \notin \Omega_T^{\text{opt}}) < 1$.

We denote $\bar{P}_t = \prod_{i=2}^{t} P(T_i \notin \Omega_T^{\text{opt}} | T_{i-1} \notin \Omega_T^{\text{opt}})$; then, $\lim_{t\to+\infty} \bar{P}_t = 0$. Since $P(T_{t+1} \notin \Omega_T^{\text{opt}} | T_t \in \Omega_T^{\text{opt}}) = 0$,

$P(T_{t+1} \in \Omega_T^{\text{opt}} | T_t \in \Omega_T^{\text{opt}}) = 1 - P(T_{t+1} \notin \Omega_T^{\text{opt}} | T_t \in \Omega_T^{\text{opt}}) = 1$.

According to the total probability formula, then we have

$$\begin{aligned}
&P(T_t \notin \Omega_T^{\text{opt}}) \\
&= P(T_t \notin \Omega_T^{\text{opt}} | T_{t-1} \notin \Omega_T^{\text{opt}}) P(T_{t-1} \notin \Omega_T^{\text{opt}}) \\
&\quad + P(T_t \notin \Omega_T^{\text{opt}} | T_{t-1} \in \Omega_T^{\text{opt}}) P(T_{t-1} \in \Omega_T^{\text{opt}}) \\
&= P(T_t \notin \Omega_T^{\text{opt}} | T_{t-1} \notin \Omega_T^{\text{opt}}) P(T_{t-1} \notin \Omega_T^{\text{opt}}) \\
&= P(T_1 \notin \Omega_T^{\text{opt}}) \prod_{i=1}^{t} P(T_{t+1} \notin \Omega_T^{\text{opt}} | T_t \notin \Omega_T^{\text{opt}}) \\
&= P(T_1 \notin \Omega_T^{\text{opt}}) \bar{P}_t.
\end{aligned}$$

Since $\lim_{t\to+\infty} \bar{P}_t = 0$, $\lim_{t\to+\infty} P(T_t \notin \Omega_T^{\text{opt}}) = 0$. Thus, Theorem 1 is proven

$$\lim_{t\to+\infty} P(T_t \in \Omega_T^{\text{opt}}) = 1 - \lim_{t\to+\infty} P(T_t \notin \Omega_T^{\text{opt}}) = 1.$$

APPENDIX B
PROOF OF THEOREM 2

Since $\lambda_t = P(T_t \in \Omega_T^{\text{opt}})$ and μ denotes the convergence time of the random process $\{T_t\}_{t=1}^{+\infty}$ reaching the optimal state space Ω_T^{opt}, we have $\lambda_t = P(\mu \le t)$. Then

$$P(\mu = t) = P(\mu \le t) - P(\mu \le t-1) = \lambda_t - \lambda_{t-1}.$$

Thus

$$E[\mu] = \sum_{t=1}^{+\infty} tP(\mu = t) = \sum_{t=1}^{+\infty} t(\lambda_t - \lambda_{t-1}) = \sum_{t=1}^{+\infty} (1-\lambda_t).$$

Since $\lambda_t = P(T_t \in \Omega_T^{\text{opt}}) = P(\mu \le t)$, we have

$$\begin{aligned}
\lambda_t = (1-\lambda_{t-1}) &P(T_t \in \Omega_T^{\text{opt}} | T_{t-1} \notin \Omega_T^{\text{opt}}) \\
&+ \lambda_{t-1} P(T_t \in \Omega_T^{\text{opt}} | T_{t-1} \in \Omega_T^{\text{opt}}).
\end{aligned}$$

Note that $P(T_t \in \Omega_T^{\text{opt}} | T_{t-1} \in \Omega_T^{\text{opt}}) = 1$; then

$$\lambda_t = (1-\lambda_{t-1}) P(T_t \in \Omega_T^{\text{opt}} | T_{t-1} \notin \Omega_T^{\text{opt}}) + \lambda_{t-1}.$$

Thus

$$\begin{aligned}
1-\lambda_t &= 1-\lambda_{t-1} - (1-\lambda_{t-1}) P(T_t \in \Omega_T^{\text{opt}} | T_{t-1} \notin \Omega_T^{\text{opt}}) \\
&= \left(1 - P(T_t \in \Omega_T^{\text{opt}} | T_{t-1} \notin \Omega_T^{\text{opt}})\right)(1-\lambda_{t-1}) \\
&= (1-\lambda_1) \prod_{2}^{t} \left(1 - P(T_t \in \Omega_T^{\text{opt}} | T_{t-1} \notin \Omega_T^{\text{opt}})\right).
\end{aligned}$$

Thus

$$\begin{aligned}
E[\mu] &= \sum_{t=1}^{+\infty} (1-\lambda_t) \\
&= \sum_{t=1}^{+\infty} \left[(1-\lambda_1) \prod_{1}^{t} \left(1 - P(T_t \in \Omega_T^{\text{opt}} | T_{t-1} \notin \Omega_T^{\text{opt}})\right) \right].
\end{aligned} \tag{13}$$

Here, we define the optimal trajectory as $\tau^* = \{s_1^*, a_1^*, s_2^*, a_2^*, \ldots, s_h^*, a_h^*\}$.

Supplemented by ϵ-greedy, the sampling policy is expressed as

$$\pi^t(a|s) = \begin{cases} 1-\epsilon+\epsilon/|A|, & \text{if } a = A^* \\ \epsilon/|A|, & \text{if } a \ne A^* \end{cases} \tag{14}$$

where ϵ denotes the exploration rate, $|A|$ denotes the size of action space, and A^* denotes the optimal action.

We denote $P(\tau^*) = \prod_{j=0}^{h} P(s_{j+1}^* | s_j^*, a_j^*)$; then, $P(T_t \in \Omega_T^{\text{opt}} | T_{t-1} \notin \Omega_T^{\text{opt}}) = \prod_{j=0}^{h} \pi^t(a_j^* | s_j^*) P(s_{j+1}^* | s_j^*, a_j^*) = P(\tau^*) \prod_{j=0}^{h} \pi^t(a_j^* | s_j^*)$.

Based on Eqn (14), we have

$$\begin{aligned}
P(\tau^*)\left[\frac{\epsilon}{|A|}\right]^h &\le P(T_t \in \Omega_T^{\text{opt}} | T_{t-1} \notin \Omega_T^{\text{opt}}) \\
&\le P(\tau^*)\left[1-\epsilon+\frac{\epsilon}{|A|}\right]^h.
\end{aligned} \tag{15}$$

Combining (13) and (15), we have

$$\begin{aligned}
(1-\lambda_1) &\sum_{t=1}^{+\infty} \prod_{1}^{t} \left\{1 - P(\tau^*)\left[1-\epsilon+\frac{\epsilon}{|A|}\right]^h\right\} \\
&\le E[\mu] \le (1-\lambda_1) \sum_{t=1}^{+\infty} \prod_{1}^{t} \left\{P(\tau^*)\left[\frac{\epsilon}{|A|}\right]^h\right\}.
\end{aligned}$$

Based on the summation rule of geometric progression, we have

$$\begin{aligned}
(1-\lambda_1) &\left\{P(\tau^*)\left[1-\epsilon+\frac{\epsilon}{|A|}\right]^h\right\}^{-1} \\
&\le E[\mu] \le (1-\lambda_1) \left\{P(\tau^*)\left[\frac{\epsilon}{|A|}\right]^h\right\}^{-1}.
\end{aligned}$$

Thus, Theorem 2 is proven.

APPENDIX C
EXPERIMENTAL DETAILS

A. Charging and Delivery

In the game of charging and delivery, the packages appear in the following coordinate positions in sequence: [1, 4], [4, 0], [2, 0], [4, 4], [0, 2], [4, 3], [0, 0], [4, 2], [2, 4], [4, 1], [3, 3], and [2, 0]. The initial position of the agent is [0, 0]. The state is represented by a 1-D vector, and each element, in turn, represents the position of the agent, the remaining power, the position of the current package, and the number of times the agent has been charged. The policy network consists of three fully connected layers, and the size of input–output in each layer is 4∗100, 100∗50, and 50∗4. The weight network is similar to the policy network, except that the action is extended into the state. The capacity of the trajectory buffer is set to $M = 300$. Each iteration retains top $K = 150$ preference optimal trajectories in the current buffer.

B. Doom Game

Like in most previous approaches [10], [45], the frame-skip technique was used in this experiment. In our setting, the agent can only receive a screen input every four frames, and the action is performed repeatedly over all the skipped frames. This can not only speed up the training but also improve the performance of the policy. Considering that states are partially observable, the output of the convolutional neural network is given to an LSTM layer [46]. However, it is still difficult to extract key information from high-dimensional visual input. Thus, some game features are directly extracted from the game engine as extended input to the network. These features include whether to pick up items or kill enemies on the current frame. This has been proven by previous methods [10] that can greatly improve the performance of policy learning. The capacity of the trajectory buffer is set to $M = 500$. Each iteration retains top $K = 150$ preference optimal trajectories in the current buffer.

REFERENCES

[1] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[2] V. Mnih *et al.*, "Playing Atari with deep reinforcement learning," 2013, *arXiv:1312.5602*. [Online]. Available: http://arxiv.org/abs/1312.5602

[3] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*. [Online]. Available: http://arxiv.org/abs/1509.02971

[4] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *Int. J. Robot. Res.*, vol. 37, nos. 4–5, pp. 421–436, Apr. 2018.

[5] J. A. Arjona-Medina, M. Gillhofer, M. Widrich, T. Unterthiner, J. Brandstetter, and S. Hochreiter, "RUDDER: Return decomposition for delayed rewards," 2018, *arXiv:1806.07857*. [Online]. Available: http://arxiv.org/abs/1806.07857

[6] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Proc. ICML*, vol. 99, 1999, pp. 278–287.

[7] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jul. 2017, pp. 16–17.

[8] M. Andrychowicz *et al.*, "Hindsight experience replay," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5048–5058.

[9] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, "ViZ-Doom: A doom-based AI research platform for visual reinforcement learning," in *Proc. IEEE Conf. Comput. Intell. Games (CIG)*, Sep. 2016, pp. 1–8.

[10] G. Lample and D. S. Chaplot, "Playing FPS games with deep reinforcement learning," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 1–7.

[11] I. Popov *et al.*, "Data-efficient deep reinforcement learning for dexterous manipulation," 2017, *arXiv:1704.03073*. [Online]. Available: http://arxiv.org/abs/1704.03073

[12] J. Randløv and P. Alstrøm, "Learning to drive a bicycle using reinforcement learning and shaping," in *Proc. ICML*, vol. 98, 1998, pp. 463–471.

[13] C. Li and K. Czarnecki, "Urban driving with multi-objective deep reinforcement learning," in *Proc. 18th Int. Conf. Auton. Agents MultiAgent Syst.*, 2019, pp. 359–367.

[14] T. Tajmajer, "Modular multi-objective deep reinforcement learning with decision values," in *Proc. Federated Conf. Comput. Sci. Inf. Syst. (FedCSIS)*, Sep. 2018, pp. 85–93.

[15] C. Liu, X. Xu, and D. Hu, "Multiobjective reinforcement learning: A comprehensive overview," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 45, no. 3, pp. 385–398, Mar. 2015.

[16] P. Vamplew, R. Dazeley, A. Berry, R. Issabekov, and E. Dekker, "Empirical evaluation methods for multiobjective reinforcement learning algorithms," *Mach. Learn.*, vol. 84, nos. 1–2, pp. 51–80, Jul. 2011.

[17] L. Barrett and S. Narayanan, "Learning all optimal policies with multiple criteria," in *Proc. 25th Int. Conf. Mach. Learn. (ICML)*, 2008, pp. 41–47.

[18] C. R. Shelton, "Balancing multiple sources of reward in reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2000, pp. 1082–1088.

[19] F. Pianosi, A. Castelletti, and M. Restelli, "Tree-based fitted Q-iteration for multi-objective Markov decision processes in water resource management," *J. Hydroinform.*, vol. 15, no. 2, pp. 258–270, Apr. 2013.

[20] A. D. Laud, "Theory and application of reward shaping in reinforcement learning," Univ. Illinois Urbana-Champaign, Champaign, IL, USA, 2004.

[21] S. Devlin, D. Kudenko, and M. Grześ, "An empirical study of potential-based reward shaping and advice in complex, multi-agent systems," *Adv. Complex Syst.*, vol. 14, no. 2, pp. 251–278, 2011.

[22] M. Jaderberg *et al.*, "Reinforcement learning with unsupervised auxiliary tasks," in *Proc. ICLR*, 2016, pp. 1–14.

[23] M. Grzes and D. Kudenko, "Plan-based reward shaping for reinforcement learning," in *Proc. 4th Int. IEEE Conf. Intell. Syst.*, vol. 2, Sep. 2008, pp. 10–22.

[24] Z. Gábor, Z. Kalmár, and C. Szepesvári, "Multi-criteria reinforcement learning," in *Proc. ICML*, vol. 98, 1998, pp. 197–205.

[25] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proc. 21st Int. Conf. Mach. Learn.*, 2004, p. 1.

[26] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 4299–4307.

[27] D. S. Brown, W. Goo, P. Nagarajan, and S. Niekum, "Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations," 2019, *arXiv:1904.06387*. [Online]. Available: http://arxiv.org/abs/1904.06387

[28] C. Wirth, R. Akrour, G. Neumann, and J. Fürnkranz, "A survey of preference-based reinforcement learning methods," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 4945–4990, 2017.

[29] D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley, "A survey of multi-objective sequential decision-making," *J. Artif. Intell. Res.*, vol. 48, pp. 67–113, Oct. 2013.

[30] K. V. Moffaert, M. M. Drugan, and A. Nowé, "Scalarized multi-objective reinforcement learning: Novel design techniques," in *Proc. IEEE SSCI*, Apr. 2013, pp. 191–199.

[31] P. Geibel, "Reinforcement learning with bounded risk," in *Proc. ICML*, 2001, pp. 162–169.

[32] R. Issabekov and P. Vamplew, "An empirical comparison of two common multiobjective reinforcement learning algorithms," in *Proc. Australas. Joint Conf. Artif. Intell.* Berlin, Germany: Springer, 2012, pp. 626–636.

[33] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," *IEEE Trans. Neural Netw.*, vol. 9, no. 5, p. 1054, Sep. 1998.

[34] Z. Xu, H. Van Hasselt, and D. Silver, "Meta-gradient reinforcement learning," 2018, *arXiv:1805.09801*. [Online]. Available: https://arxiv.org/abs/1805.09801

[35] D. V. Prokhorov and D. C. Wunsch, "Adaptive critic designs," *IEEE Trans. Neural Netw.*, vol. 8, no. 5, pp. 997–1007, Sep. 1997.

[36] K. Van Moffaert, M. M. Drugan, and A. Nowé, "Scalarized multi-objective reinforcement learning: Novel design techniques," in *Proc. IEEE Symp. Adapt. Dyn. Program. Reinforcement Learn. (ADPRL)*, Apr. 2013, pp. 191–199.

[37] P. Vamplew, J. Yearwood, R. Dazeley, and A. Berry, "On the limitations of scalarisation for multi-objective reinforcement learning of Pareto fronts," in *Proc. Australas. Joint Conf. Artif. Intell.* Berlin, Germany: Springer, 2008, pp. 372–378.

[38] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

[39] Y. Wu and Y. Tian, "Training agent for first-person shooter game with actor-critic curriculum learning," in *Proc. ICLR*, 2017.

[40] S. Russell, "Should we fear supersmart robots?" *Sci. Amer.*, vol. 314, no. 6, pp. 58–59, May 2016.

[41] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in AI safety," 2016, *arXiv:1606.06565*. [Online]. Available: http://arxiv.org/abs/1606.06565

[42] T. D. Kulkarni, K. R. Narasimhan, A. Saeedi, and J. B. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *Proc. 30th Int. Conf. Neural Inf. Process. Syst. (NIPS)*, 2016, pp. 3682–3690.

[43] P.-L. Bacon, J. Harb, and D. Precup, "The option-critic architecture," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 1726–1734.

[44] H. Chen *et al.*, "Large-scale interactive recommendation with tree-structured policy gradient," in *Proc. AAAI Conf. Artif. Intell.*, 2019, vol. 33, no. 1, pp. 3312–3320.

[45] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *J. Artif. Intell. Res.*, vol. 47, pp. 253–279, Jun. 2013.

[46] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

Jinsheng Ren received the B.S. degree in automation from the University of Electronic Science and Technology of China, Chengdu, China, in 2017. He is currently pursuing the Ph.D. degree with the Department of Automation, Tsinghua University, Beijing, China.

His current research interests include computer vision, artificial intelligence, lifelong learning, reinforcement learning, and learning theory.

Shangqi Guo received the B.S. degree in mathematics and physics basic science from the University of Electronic Science and Technology of China, Chengdu, China, in 2015. He is currently pursuing the Ph.D. degree with the Department of Automation, Tsinghua University, Beijing, China.

His current research interests include inference in artificial intelligence, brain-inspired computing, computational neuroscience, and reinforcement learning.

Feng Chen (Member, IEEE) received the B.S. and M.S. degrees in automation from Saint Petersburg Polytechnic University, Saint Petersburg, Russia, in 1994 and 1996, respectively, and the Ph.D. degree from the Automation Department, Tsinghua University, Beijing, China, in 2000. He is currently a Professor with Tsinghua University. His current research interests include artificial general intelligence, brain-inspired computing, and reinforcement learning.