Information Sciences 553 (2021) 172–188

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Recruitment-imitation mechanism for evolutionary reinforcement learning

Shuai Lü [a,b,*], Shuai Han [a,b], Wenbo Zhou [a,b], Junwei Zhang [a,b]

[a] Key Laboratory of Symbolic Computation and Knowledge Engineering (Jilin University), Ministry of Education, Changchun 130012, China
[b] College of Computer Science and Technology, Jilin University, Changchun 130012, China

ARTICLE INFO

Article history:
Received 10 December 2019
Received in revised form 7 December 2020
Accepted 8 December 2020
Available online 15 December 2020

Keywords:
Evolutionary reinforcement learning
Reinforcement learning
Evolutionary algorithms
Imitation learning

ABSTRACT

Reinforcement learning, evolutionary algorithms and imitation learning are three principal methods to deal with continuous control tasks. Reinforcement learning is sample efficient, yet sensitive to hyperparameters settings and needs efficient exploration; Evolutionary algorithms are stable, but with low sample efficiency; Imitation learning is both sample efficient and stable, however it requires the guidance of expert data. In this paper, we propose Recruitment-imitation Mechanism (RIM) for evolutionary reinforcement learning, a scalable framework that combines advantages of the three methods mentioned above. The core of this framework is a dual-actors and single critic reinforcement learning agent. This agent can recruit high-fitness actors from the population performing evolutionary algorithms, which instructs itself to learn from experience replay buffer. At the same time, low-fitness actors in the evolutionary population can imitate behavior patterns of the reinforcement learning agent and promote their fitness level. Reinforcement and imitation learners in this framework can be replaced with any off-policy actor-critic reinforcement learner and data-driven imitation learner. We evaluate RIM on a series of benchmarks for continuous control tasks in Mujoco. The experimental results show that RIM outperforms prior evolutionary or reinforcement learning methods. The performance of RIM's components is significantly better than components of previous evolutionary reinforcement learning algorithm, and the recruitment using soft update enables reinforcement learning agent to learn faster than that using hard update.

1. Introduction

An important goal of artificial intelligence is to develop agents with excellent decision-making capabilities in complex and uncertain environments. In recent years, the rapid development of deep neural network enables agents based on reinforcement learning methods to perform well in complex control tasks [1–6]. However, reinforcement learning methods cannot always handle reward sparse problems effectively and their parameters are very sensitive to disturbances. Recent studies have shown that evolutionary algorithms are better at dealing with sparse reward environments [7], and can be employed as an extensible alternative to reinforcement learning in various tasks [8]. When a specific control task has expert data as a guide, imitation learning can also train agents efficiently. Currently, methods based on imitation learning have been successfully applied to drones [9], automated driving [10], and other fields.

* Corresponding author at: Key Laboratory of Symbolic Computation and Knowledge Engineering (Jilin University), Ministry of Education, Changchun 130012, China.

E-mail address: lus@jlu.edu.cn (S. Lü).

https://doi.org/10.1016/j.ins.2020.12.017
0020-0255/© 2020 Elsevier Inc. All rights reserved.

These artificial intelligence algorithms have close relationships with principles of biology [11]. Evolutionary algorithms are inspired by evolution of species. In each generation of the population, there are plenty of random mutations in different individuals. Individuals with positive mutations will be selected by the environment. In imitation learning, individuals with poor fitness can learn behavior patterns from individuals with good fitness in a supervised learning way. At the same time, the diversity of genes (parameters) is preserved. The biological basis of reinforcement learning is that individuals learn correct or wrong actions through trials and errors from their interactions with the environment. Off-policy reinforcement learning approaches allow individuals to learn and update their policies from historical data [1,12–14]. If reinforcement learning individuals are allowed to learn from the entire historical interactions between population and environment, and regularly copy reinforcement learning individuals into the population to participate in the evolution, both the learning process of reinforcement learning and the evolutionary process of evolutionary algorithms can be accelerated [15,16]. However, there are still two problems.

- Reinforcement learning agents can only learn from the experience, but not directly accept the guidance of elites in the current population.
- Reinforcement learning agents must have the same structure as individuals in the population, or at least have a similar parameterized form. Otherwise, reinforcement learning individuals that are copied into the population may not be able to participate in the evolutionary process.

In response to the problems mentioned above, this paper proposes Recruitment-imitation Mechanism (RIM) for evolutionary reinforcement learning. The framework of RIM for evolutionary reinforcement learning is presented in Fig. 1.

The recruitment process allows reinforcement learning (RL) agent to recruit the best individual from population to participate in RL agent decision and learning process. This requires RL agent to have the structure of Fig. 2.

In Fig. 2, gradient policy network and recruitment policy network simultaneously accept the current state as input, and respectively produce actions as output. Critic network compares the potential rewards of actions produced by the two policy networks, and outputs the action with higher reward. The dual-policy decision-making mechanism not only enables the RL agent to produce better experiences, but also enables the RL agent to better estimate Q value, which enables the learning process of RL agent to directly accept the guidance of excellent policy in the population. The imitation process permits low-fitness individuals in the population to learn behavioral patterns of RL agent. Since the structure of RL agent is inconsistent with that of individuals in the population, RL agent cannot be directly injected into the population and participate in the evolution. The imitation process is designed for addressing this problem.

RIM is more competitive than previous methods. First of all, RIM almost inherits all advantages of evolutionary reinforcement learning (ERL), for example, more efficient exploration than RL and higher sample efficiency than evolutionary algorithms (EAs) [15]. Secondly, RIM employs an imitation learning (IL) method, but does not require the guidance of external expert data compared with a single IL algorithm. Finally, RIM has higher sample efficiency and better asymptotic performance than ERL. Compared with ERL, the dual-policy structure in RIM enables the reinforcement learning agent to make better decisions even when the learning is insufficient. Moreover, the imitation learning process of individuals in the RIM population directly improves the performance of the population, and it also strengthens the exploration of the parameter space during evolution.

The main contributions of this paper are as follows:

- We propose Recruitment-imitation Mechanism (RIM) and an evolutionary reinforcement learning framework that applies this mechanism. At the core of RIM, a dual-policy RL structure is designed, which allows critic network to determines actions.

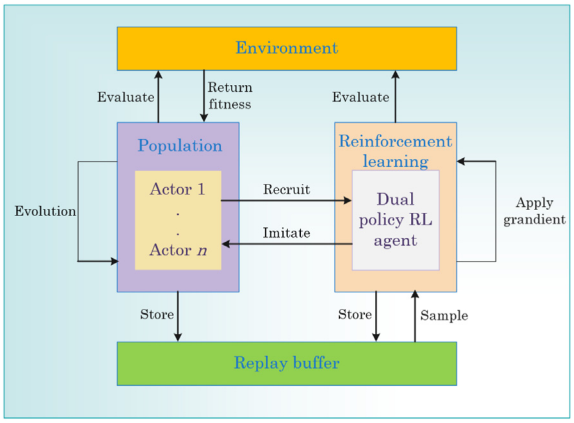


Fig. 1. Framework of RIM for evolutionary reinforcement learning.

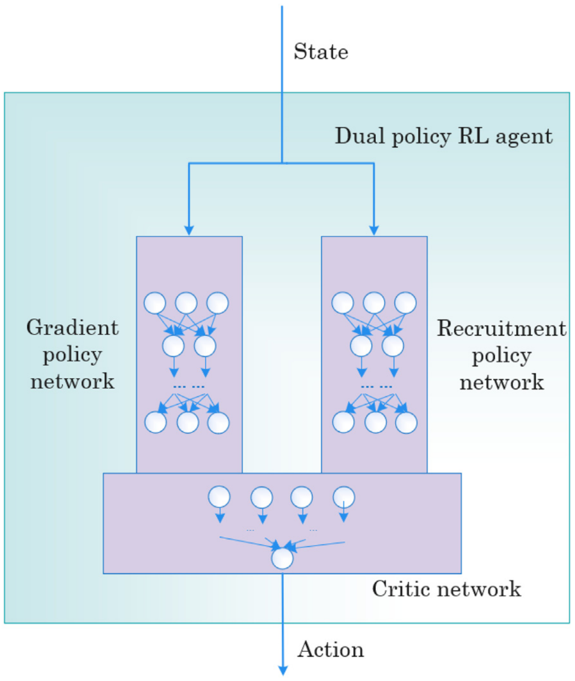


Fig. 2. Structure of dual policy RL agent.

- We present a series of optimization techniques for RIM, including an off-policy imitation learning algorithm that directly uses the experience replay buffer and soft updating strategies for recruiting networks.
- We compare the performance of RIM with that of previous algorithms on Mujoco benchmarks. Moreover, we discuss the impact of optimization techniques on RIM's performance.

The rest of this paper is organized as follows. Section 2 briefly introduces the background. Section 3 presents the proposed recruitment-imitation mechanism in detail. Section 4 shows the experimental results and comparisons. Section 5 discusses advantages of RIM and outlines the future work.

2. Related work

Combining reinforcement learning with imitation learning or evolutionary algorithms is not a new idea [17–21]. The traditional combination methods mostly consider imitation learning or evolutionary algorithms as a sub-process of reinforcement learning. Typical practices in such methods include: leveraging the distribution estimation method to improve the exploration noise in reinforcement learning [22], utilizing the evolutionary method to optimize the approximation function in *Q* learning [23], or using the imitation learning to optimize the initial stage of the RL process [24]. Unlike these traditional practices, evolution and imitation learning in RIM are not sub-processes of reinforcement learning. In RIM, evolution and reinforcement learning are two relatively independent learning processes, while imitation learning is a way of synchronizing the behavioral policy of dual-policy agents with certain individuals in populations.

Evolutionary Reinforcement Learning (ERL) [15] provides a new paradigm for the combination of evolutionary algorithms and reinforcement learning. ERL's approach is to reuse the interaction data between the population and the environment, and inject RL policy into the population. RIM can be viewed as an extension of ERL. The extended parts include:

- Combining evolutionary algorithms from the perspective of reinforcement learning. Namely, recruiting policy directly from the population is used to participate in reinforcement learning processes.
- Constructing a dual-policy agent in the RL component. The agent can integrate two policies to generate experience, and give better estimation of *Q* value when policy learning is insufficient.
- Leveraging off-policy imitation learning to synchronize the behavioral policy of dual-policy agent into the population. By this way, the RL agent does not have to maintain the same structure as the individuals in the population.

In the latest works, there are other ways to extend ERL: Collaborative Evolutionary Reinforcement Learning (CERL) [25] uses different combinations of learners to complete exploration in different time-horizons of tasks; Cross-entropy Method Reinforcement Learning (CEM-RL) [16] replaces the combination of standard evolutionary algorithm and DDPG [1] in ERL with a combination of Cross-entropy Method (CEM) and Twin Delayed Deep Deterministic policy gradient (TD3) [26]. Unlike the works mentioned above, RIM is neither an optimization for task exploration nor a replacement of sub-components in ERL. Instead, it introduces a recruitment imitation mechanism that enables reinforcement learning, imitation learning, and evolutionary algorithms to be more effectively combined. In other words, RIM, CERL and CEM-RL are independent optimizations in different directions of ERL.

3. Background

This section introduces the background of deep reinforcement learning, evolutionary algorithms and imitation learning.

3.1. Deep reinforcement learning

Reinforcement learning methods abstract the interaction between an agent and environments as a Markov decision process. At each discrete time step t, the environment provides an observation s_t to the agent, and the agent takes an action a_t as a response to this observation. Then, the environment returns a reward r_t and the next state s_{t+1} to the agent. The goal of reinforcement learning is to maximize the expectation of $R_t = \sum_{k=0}^{K} \gamma^k r_{t+k}$, where R_t indicates the sum of cumulative discounted rewards of K steps from the current moment. $\gamma \in (0, 1]$ is a discount factor.

Deep deterministic policy gradient (DDPG) is a widely used model-free reinforcement learning algorithm based on actor-critic structure [1]. In DDPG, actor and critic are parameterized as $\pi(s|\theta^{\pi})$ and $Q(s, a|\theta^{Q})$ respectively. They are called current networks. In addition, replications of actor and critic networks $\pi'(s|\theta^{\pi'})$ and $Q'(s, a|\theta^{Q'})$ are used to provide consistent targets for the learning process. They are called target networks. During the learning process, target networks will be soft updated based on current networks and a weighting factor τ. DDPG completes off-policy learning by sampling experiences from a repaly buffer R. That is, for each interaction between an agent and environments, the tuple (s_t, a_t, r_t, s_{t+1}) is stored into the replay buffer. One advantage of off-policy reinforcement learning is the sample efficiency of these algorithms. The other advantage is that learning from historical data can decouple exploration process and learning process. Exploration policy of DDPG is constructed from actor policy and noise: $\pi_e(s_t) = \pi(s_t|\theta^{\pi}) + \mathcal{N}$. The actor and critic networks are updated by sampling mini-batch experiences from the replay buffer. Critic network is updated by minimizing the loss function as follows:

$$L = \frac{1}{N}\sum_i \left(y_t^{(i)} - Q(s_t^{(i)}, a_t^{(i)}|\theta^Q)\right)^2 \tag{1}$$

where $y_t^{(i)} = r_t^{(i)} + \gamma Q'(s_{t+1}^{(i)}, \pi'(s_{t+1}^{(i)}|\theta^{\pi'})|\theta^{Q'})$. The actor network is updated according to sampled policy gradient:

$$\nabla_{\theta^{\pi}} \pi|_{s^{(i)}} \approx \frac{1}{N}\sum_i \nabla_a Q(s^{(i)}, a|\theta^Q)|_{a=\pi(s^{(i)})} \nabla_{\theta^{\pi}} \pi(s^{(i)}|\theta^{\pi}) \tag{2}$$

3.2. Evolutionary algorithms

Evolutionary algorithms (EAs) [27,28] are black-box optimization algorithms that are inspired by natural evolutionary processes. The evolutionary process acts on a population composed of several parameterized candidate solutions (individuals). During each iteration, parameters of individuals in the population are randomly perturbed (mutated), enabling new individuals to be generated. After generating new individuals, the environment evaluates their fitness. Individuals with higher fitness have higher probability to be selected, and those selected individuals will participate in the next iterative process. When evolutionary algorithms are implemented to solve control tasks, several actor networks will participate in the evolutionary process. During the iterative process, actor networks with higher fitness will be retained as elites and shielded from mutation step [15]. EAs can be employed as an alternative for reinforcement learning techniques based on Markov decision processes [7]. Furthermore, in this paper, the fitness of individuals in the population is defined as the average rewards that individuals obtain in the environment, which is consistent with the previous methods [15,16,25].

3.3. Imitation learning

In imitation learning, an actor imitates expert behaviors to obtain a policy that is close to expert performance. Dataset aggregation (DAgger) method [29] is a widely used imitation learning algorithm. It is an iterative reduction to online policy training methods. Suppose that there is an expert data set $D = \{(s_1, a_1), \dots, (s_n, a_n)\}$. During each iteration, the actor is trained in this dataset in a supervised manner. The loss function during training can be denoted as: $J(\theta) = \frac{1}{K}\sum_{k=1}^{K} L(\pi_{il}(s_k), a_k)$, where π_{il} is a policy to be trained and K is the batch size of samples. After convergence, policy π_{il} is carried out to generate a state access set $D_{\pi} = \{s_{n+1}, \dots, s_{n+m}\}$. Then D_{π} is labeled by actions output from the expert policy, and the expert data is accumulated: $D \leftarrow D \cup D_{\pi}$. Then proceed to the next iteration. The advantage of DAgger is that it can employ expert policy to supervise actors to recover from errors. DAgger is a class of Follow-The-Leader algorithms [30].

4. Recruitment-imitation mechanism

Recruitment-imitation mechanism in evolutionary reinforcement learning is presented in this section.

4.1. Dual policy reinforcement learning agent

The core of recruitment-imitation mechanism is a dual policy RL agent with dual-actors and single-critic. In the iterative process of the evolutionary reinforcement learning algorithm, the dual policy RL agent can recruit excellent individuals from the population to participate in decision-making or reinforcement learning process. In addition, individuals with poor performance in the population can periodically imitate behavior patterns of dual policy RL agent to accelerate evolutionary speed of the population.

Dual policy RL agent in recruitment-imitation mechanism still uses the actor-critic structure but has two actor networks, including a gradient policy network $\pi_{pg}(s_t|\theta^{pg})$ and a recruitment policy network $\pi_{ea}(s_t|\theta^{ea})$. When the agent makes decisions, the critic network will identify which policy under the current state s_t is potentially more profitable. The policy of the dual policy RL agent can be indicated as:

$$\pi_{rl} = \begin{cases} \pi_{pg}, & Q(s_t, \pi_{pg}(s_t|\theta^{pg})|\theta^Q) \geqslant Q(s_t, \pi_{ea}(s_t|\theta^{ea})|\theta^Q) \\ \pi_{ea}, & \mathit{otherwise} \end{cases} \tag{3}$$

where π_{rl} indicates the overall behavior policy of the dual policy RL agent. Similar to DDPG, during the learning process, the critic network is updated by minimizing the loss function of Eq. (1), except that $y_t^{(i)}$ is estimated using π_{rl}:

$$y_t^{(i)} = r_t^{(i)} + \gamma Q'(s_{t+1}^{(i)}, \pi_{rl}(s_{t+1}^{(i)})|\theta^{Q'}) \tag{4}$$

The gradient policy network is still updated with the sampled policy gradient according to Eq. (2).

There are two reasons that the dual policy RL agent can make RIM perform better:

- The dual policy RL agent can generate better experiences according to the behavior pattern in Eq. (3).
- When gradient update is performed, π_{rl} used in Eq. (4) can make more accurate estimation of Q'.

Then we prove that Eq. (4) can give more accurate estimation of Q'.

Theorem 1. *Suppose that Q' and Q converge at the same position for policy π^*. The converged policy to maximize Q' or Q is π^*, and the estimation value using π^* is Q^*. The mean of Q' estimation of DDPG is denoted as $E_{ddpg}(\hat{Q}')$ and the mean of Q' estimation of RIM is denoted as $E_{rim}(\hat{Q}')$. Then, $E_{ddpg}(\hat{Q}') \leqslant E_{rim}(\hat{Q}') \leqslant Q^*$.*

Proof. Since π^* is the policy that maximizes Q' (i.e., Q^*), any policy that is not π^* will cause the estimate of Q' to be less than Q^*. So, there is $E_{rim}(\hat{Q}') \leqslant Q^*$. $E_{rim}(\hat{Q}') = Q^*$ if and only if the policy of π_{rl} in Eq. (3) is always π^*. Also there is $E_{ddpg}(\hat{Q}') \leqslant Q^*$. $E_{ddpg}(\hat{Q}') = Q^*$ if and only if the policy π_{ddpg} of DDPG is always π^* .

Assume that DDPG's behavior policy π_{ddpg} causes the estimation of Q' to be shifted downward by x with the probability of $p(x)$, then $E_{ddpg}(\hat{Q}') = \int_{-\infty}^{+\infty}(Q^* - x)p(x)dx$. Since RIM uses the policy in Eq. (3) to estimate Q' in Eq. (4). When π_{pg} (i.e., π_{ddpg}) estimates Q' downwards by x with the probability of $p(x)$, π_{ea} can upwards correct y ($y \geqslant 0$) with the probability of $q(y|x)$. Then we have:

$$\begin{aligned} E_{rim}(\hat{Q}') &= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}(Q^* - x + y)q(y|x)p(x)dydx \\ &= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}(Q^* - x)q(y|x)p(x)dydx + \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} yq(y|x)p(x)dydx \\ &= \int_{-\infty}^{+\infty}(Q^* - x)(\int_{-\infty}^{+\infty} q(y|x)dy)p(x)dx + \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} yq(y|x)p(x)dydx \\ &= \int_{-\infty}^{+\infty}(Q^* - x)p(x)dx + \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} yq(y|x)p(x)dydx \\ &= E_{ddpg}(\hat{Q}') + \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} yq(y|x)p(x)dydx \end{aligned}$$

Considering $y \geqslant 0$ and $q(y|x)p(x) \geqslant 0$, then $\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} yq(y|x)p(x)dydx \geqslant 0$, so it holds that $E_{ddpg}(\hat{Q}') \leqslant E_{rim}(\hat{Q}')$. □

In Theorem 1, we can assume that Q and Q' converge at the same position because Q and Q' are very close due to the soft update setup in continuous domains [26]. Theorem 1 states that the estimation of Q' using π_{rim} is more accurate than that using π_{ddpg}, or closer to Q^*. Unlike overestimation problems [31,32], Theorem 1 states that when π_{ddpg} does not converge sufficiently during the updating process according to Eq. (2), if the policy π_{ea} in population gives better actions, the Q' estimation of RIM will be more accurate. This is one of the reasons that the RL component of RIM can learn better.

RIM recruits individuals from the population to participate in the parameter update of the RL agent. Another popular reinforcement learning method, Asynchronous Advantage Actor-Critic (A3C), also obtains information from agents in different processes to update global parameters [2]. However, the main difference between RIM and A3C lies in the process of obtaining the gradient. RIM uses the critic to evaluate and choose actions from the gradient policy and the recruited policy, then calculates the target value in Eq. (4) based on actions chosen by the critic and further calculates the gradient. While A3C obtains parameter information directly from agents in different processes and updates global parameters asynchronously.

4.2. Off-policy imitation learning

Due to the inconsistencies in structures of a dual policy RL agent (dual-actors and single-critic) and an individual (single-actor) in the population, the dual policy RL agent cannot be directly injected into the population to participate in the evolution. In this case, an ideal solution is to use imitation learning to unify behavior patterns of the RL agent and certain individuals in the population.

As a typical imitation learning algorithm, DAgger can get better performance with only a few iterations when the amount of data is large. Evolutionary reinforcement learning usually has a huge experience replay buffer for storing historical data. The actors in the population to be trained can directly sample states and actions in the experience replay buffer and be trained according to the following loss function:

$$J_0(\theta^{wt}) = \frac{1}{K}\sum_{k=1}^{K} L(\pi_{wt}(s_k|\theta^{wt}), \pi_{rl}(s_k)) \tag{5}$$

where π_{wt} indicates the policy of an actor to be trained. π_{wt} is usually a policy of an actor with the worst performance in the population. Fig. 3 shows the framework of off-policy imitation learning and Algorithm 1 describes the detailed process of off-policy imitation learning in RIM.

Algorithm 1: Off-policy imitation learning in RIM

```
   Input: Max epochs $M$, Replay buffer $R$, Batch size $N$, Learning rate $\alpha$
1  Get the worst policy $\pi_{wt}$ from population;
2  for $epoch = 1 \to M$ do
3      Get replay buffer size: $size \leftarrow len(R)$;
4      while $size > N$ do
5          Sample $N$ random batch $\{s_i\}$ from replay buffer;
6          Get $a_i^{wt}$ according to $\pi_{wt}(s_i|\theta^{wt})$;
7          Get $\pi_{rl}$ according to equation (3);
8          Get expert action $a_i$ according to $\pi_{rl}(s_i)$;
9          Calculate loss $L = \frac{1}{N}\sum_i |a_i^{wt} - a_i|$ between action $a_i^{wt}$ and expert
           action $a_i$;
10         Calculate $\bigtriangledown_{\theta^{wt}} L$;
11         Update $\theta^{wt} \leftarrow \theta^{wt} + \alpha \cdot \bigtriangledown_{\theta^{wt}} L$;
```

We cancel the data labeling process and accumulation process of DAgger so that the imitation process is off-policy completely. The main function of DAgger's data labeling process and accumulation process are to recover the imitation learner from errors. In the RIM environment setting, the RL agent uses a large amount of experiences generated by individuals in EA. So we tested the best, average, and the worst performance of individuals in EA, as shown in the Fig. 4. From the individual performance of individual EA, it can be inferred that in the population of RIM, there will be a large amount of erroneous data in the experience replay buffer, and most of the states sampled from the experience replay buffer are generated by a sub-optimal or wrong policy. The performance of individuals in the population varies widely, and even the interaction information of the worst individual is stored into the replay buffer.

Therefore the training process using Eq. (5) is also a process to supervise actors to recover from an error state.

4.3. Learning and evolution process of RIM

The core idea of RIM is to: 1) accelerate the learning process by recruiting elite individuals in the population to guide the policy gradient network learning, and 2) accelerate the evolutionary process by enabling individuals with poor performance in the population to imitate behavioral patterns of the dual policy RL agent.

The process of RIM is as follows:

(1) Some actor networks in the population and actor and critic networks in the dual policy agent are initialized with random weights.

(2) In the evolution process, the environment evaluates the fitness of actors in the population according to the average rewards earned by the actors in the environment and selects a part of the actors to survive with a probability according to the fitness.
(3) The actors are perturbed by mutation and crossover to generate the next generation of actors.
(4) After generating the next generation, the dual policy RL agent recruits an actor with the highest fitness in the population, copies its parameters to the recruitment policy network, and learns for a certain number of times.

Algorithm 2 describes the detailed learning process of dual policy RL agent. The worst actor in the population imitates behavioral patterns of the dual policy RL agent every several generations so that the learning results of the dual policy RL agent can be injected into the population.

Algorithm 2: Learning process of RL agent in RIM

```
Input: Max episode $M$, Batch size $N$, Learning rates $\alpha_Q$ and $\alpha_{pg}$
1  Randomly initialize critic $Q(s,a|\theta^Q)$, RL actor $\pi_{pg}(s|\theta^{pg})$ and EA actor $\pi_{ea}(s|\theta^{ea})$ ;
2  Initialize target network $Q'$, $\pi'_{pg}$, $\pi'_{ea}$ with $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{pg'} \leftarrow \theta^{pg}$, $\theta^{ea'} \leftarrow \theta^{ea}$ ;
3  Initialize replay buffer $R$, environment $Env$;
4  for $episode = 1 \rightarrow M$ do
5      Recruit champion from populations: $\theta^{ea} \leftarrow \theta^{champ}$;
6      Reset $Env$ and get a state $s$;
7      $done \leftarrow False$;
8      while $!done$ do
9          $\pi(s) \leftarrow max_\pi\{Q(s,\pi_{pg}(s)), Q(s,\pi_{ea}(s))\}$;
10         Get action $a$ according to $\pi(s) + \mathcal{N}$;
11         Execute $a$ and observe $(r, s', done) \sim Env$;
12         Store experience $(s, a, r, s')$ into replay buffer $R$;
13         Update state: $s \leftarrow s'$;
14         if $len(R) > N$ then
15             Sample $N$ random batch $\{(s_t, a_t, r_t, s_{t+1})_i\}$ from replay buffer;
16             $Q'(s_{t+1}^{(i)}, a_{t+1}^{(i)}|\theta^{Q'}) \leftarrow max\{Q'(s_{t+1}^{(i)}, \pi'_{pg}(s_{t+1}^{(i)})|\theta^{Q'}), Q'(s_{t+1}^{(i)}, \pi'_{ea}(s_{t+1}^{(i)})|\theta^{Q'})\}$;
17             $L^Q \leftarrow \frac{1}{N}\sum_i(r_t^{(i)} + Q'(s_{t+1}^{(i)}, a_{t+1}^{(i)}|\theta^{Q'}) - Q(s_t^{(i)}, a_t^{(i)}|\theta^Q))^2$;
18             $L^\pi \leftarrow -\frac{1}{N}\sum_i Q(s_t^{(i)}, \pi_{pg}(s_t^{(i)}|\theta^{pg})|\theta^Q)$;
19             Update $\theta^Q$, $\theta^{pg}$ according to $\bigtriangledown_{\theta^Q} L^Q$, $\bigtriangledown_{\theta^{pg}} L^\pi$, $\alpha_Q$ and $\alpha_{pg}$;
20             $\theta^{ea'} \leftarrow (1-\tau_0)\theta^{ea'} + \tau_0\theta^{ea}$;
21             $\theta^{Q'} \leftarrow (1-\tau_1)\theta^{Q'} + \tau_1\theta^Q$;
22             $\theta^{pg'} \leftarrow (1-\tau_1)\theta^{pg'} + \tau_1\theta^{pg}$;
```

During the evolutionary process, interaction information between actors and environments will be stored into the experience replay buffer. These experiences are not only used for the sampling process of gradient policy network learning in Algorithm 2, but also for the sampling process of the worst actor imitating in population. In the RIM setting, the structure of gradient policy network is the same as that of actors in the population, so when the imitation learning is not performed, the gradient policy network will be periodically copied into the population to accelerate the evolutionary process.

The recruitment policy network participates in the decision process of π_{rl}. Therefore, it affects the value of $y_t^{(i)}$. The learning process of critic network needs to be provided with a consistent $y_t^{(i)}$. Drawing on the idea of soft update in DDPG, we soft update the target gradient policy network and critic network. In order to further ensure consistency in the learning process, we decide to add a target recruitment policy network. Current recruitment policy network $\pi_{ea}(s_t|\theta^{ea})$ uses hard update to copy directly from the population, and the target recruitment policy network uses soft update to update parameters:

$$\theta^{ea'} \leftarrow (1-\tau_0)\theta^{ea'} + \tau_0\theta^{ea} \tag{6}$$

where $\tau_0 \ll 1$. The recruitment process using target recruitment network is called soft update in recruitment, and the recruitment process without target recruitment network is called hard update in recruitment. We will perform comparative experiments on these two recruitment methods in the experimental part.

5. Experiments

In this section, we show comparative experiments and a series of analytical results.

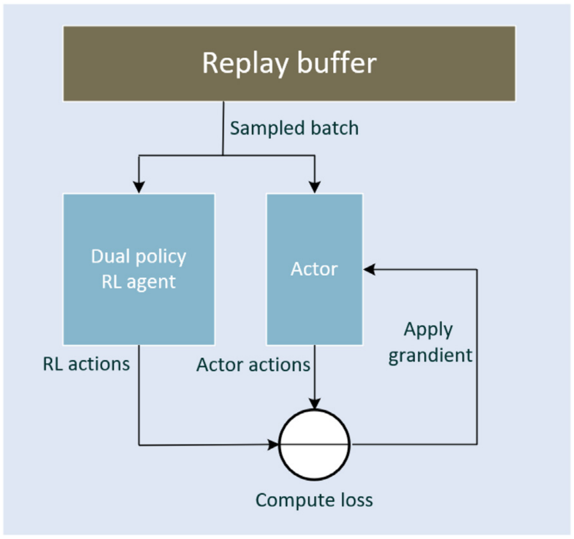


Fig. 3. Framework of off-policy imitation learning in RIM.

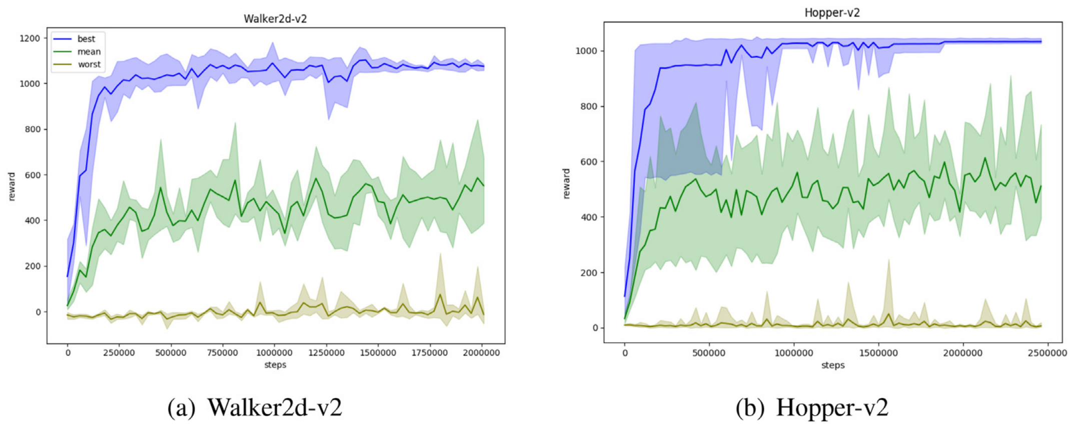


Fig. 4. Performance of different individuals in the population of EA.

5.1. Experimental settings

The experiments in this paper are performed on a computer with Nvidia GeForce GTX 1050Ti GPU and Intel(R) Core(TM) i7-8700 K CPU. RIM needs to use GPU to accelerate the training of neural networks. RIM does not need multi-process programming, so it does not need multi-core CPU.

We evaluated the performance of RIM on four continuous control tasks in Mujoco [33] hosted through the OpenAI gym [34], which are widely used in the evaluation of reinforcement learning continuous domains [35–37]. These tasks are shown in Fig. 5.

At the initial time step, the environment will provide our algorithm with a vector with a specific dimension as the initial state. At each time step, our algorithm will output an action vector with a specific dimension to the environment according to the state provided by the environment at the former time step. Then, according to the action provided by the algorithm, the environment will return a state vector and a reward. The reward is usually a scalar related to the speed of the object. The state and the reward will be the input of our algorithm at the next time step. Table 1 shows the dimensions of actions and states in all environments involved in this paper.

Baselines for comparison include Evolutionary Reinforcement Learning (ERL) algorithm [15], DDPG, and a standard evolutionary algorithm (EA). ERL is a recent algorithm which can be used to solve continuous control tasks well; DDPG is considered as one of the best reinforcement learning algorithms and is widely used in a variety of continuous control tasks. RIM and these baselines are implemented using PyTorch [38].

The reinforcement learning hyperparameters of RIM are consistent with those of ERL and DDPG. The crossover and mutation probability is 0 and 0.01 respectively in populations of RIM, ERL and EA. In the comparative experiments, RIM uses soft

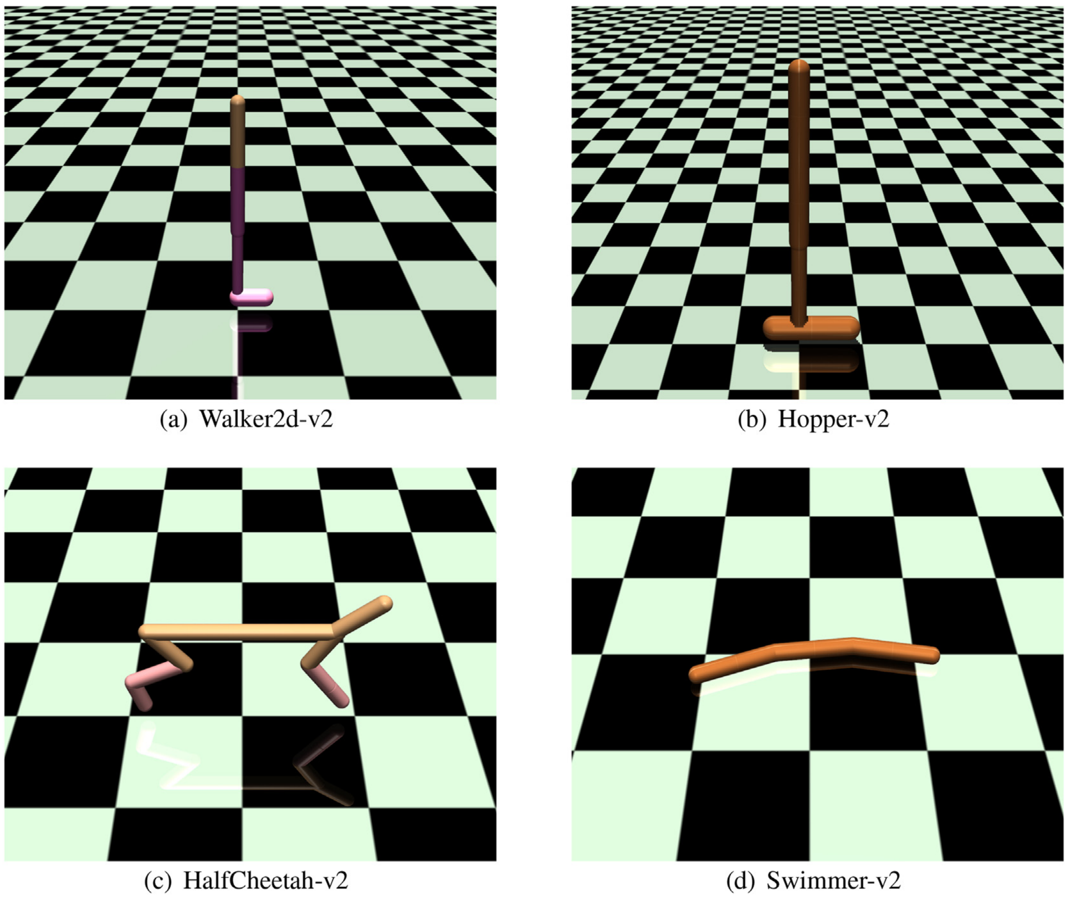


Fig. 5. Four continuous control tasks in Mujoco-based environments.

update to recruit policies from population. All actor networks and critic networks have two hidden layers, with rectified linear units (ReLU) between hidden layers. The last layer of each actor network is linked to a tanh unit. We use Adam [39] to update all network parameters. The learning rates of the gradient policy network and the critic network are 10^{-4} and 10^{-3}, respectively. The exploration noise of RL agent is generated by the Ornstein–Uhlenbeck process [40]. The number of individuals in the population is 10. Imitation learning is performed every 10 generations. The loss function of the imitation learning is least absolute error (L1), and the learning rate is 10^{-3}. The sample batch sizes of reinforcement learning and imitation learning are 128 and 32, respectively. The detailed settings of parameters for RL, EA, and IL are shown in Table 2.

The hyperparameter settings in Table 2 are common configurations in all environments [1,15]. Table 3 shows the hyperparameters with different settings in different environments. Because Walker2d-v2 and Hopper-v2 are more complex, RIM needs more interactions with the environment. Total interaction in Table 3 refers to the sum of the interactions between all components in RIM and the environment. The final fitness of an individual is the reward for a single run or the average reward over repeated runs in the environment. Number of evaluation refers to the number of runs in the environment when the final fitness for an individual is evaluated. Due to the high randomness in Walker2d-v2 and Hopper-v2, the reward obtained from a single run is not reliable, so the average reward from repeated runs is required. Elite fraction represents the proportion of elite individuals to the total population. Elite individuals will be protected from mutations during evolution. Imitation period refers to the period during which individuals perform imitation learning. Setting the imitation period to 2 or 10 means that imitation learning is performed every 2 or 10 generations. The settings for the number of evaluation and the elite fraction also refer to ERL [15].

The performance of the algorithms involved in the comparison is tested using different random seeds and is recorded every 10,000 steps. When testing, the average of five test results is recorded as the performance at this time. For DDPG, the actor's exploration noise was removed during the test. For EA, ERL and RIM, the performance of the best individuals in their population are recorded as the final performance. The scores of these algorithms are the rewards returned by environments, and the corresponding number of steps is the cumulative number of steps that an algorithm interacts with the environment as a whole. These recording methods are consistent with previous comparative experiments [15].

Table 1
Action and state dimensions of some tasks in Mujoco.

	Action dimension	State dimension
Walker2d-v2	6	17
Hopper-v2	3	11
HalfCheetah-v2	6	17
Swimmer-v2	2	8

Table 2
Detailed hyperparameter settings in RIM.

Category	Hyperparameter	Value
RL part	Hidden layers of critic networks	2
	Hidden layers of actor networks	2
	Size of hidden layers of critic networks	400
	Size of hidden layers of actor networks	300
	Activation for hidden layers of actor	ReLU
	Activation for output layer of actor	Tanh
	Activation for hidden layers of critic	ReLU
	Activation for output layer of critic	None
	Critic loss	MSE
	Optimizer	Adam
	Learning rate for critic networks	0.001
	Learning rate for actor networks	0.0001
	Batch size	128
	Discount factor	0.99
	Soft update weight	0.001
	Replay buffer size	1,000,000
	Exploration policy	Ornstein–Uhlenbeck [40]
EA part	Population size	10
	Mutation probability	0.9
	Mutation noise type	Gaussion noise
	Mutation strength	0.1
IL part	Batch size	32
	Learning rate	0.001
	Loss	L1
	Epochs	30000

Table 3
Hyperparameter settings in different environments.

Hyperparameter	Value		
	Walker2d-v2	Hopper-v2	HalfCheetah-v2 and Swimmer-v2
Total interaction	5,000,000	2,500,000	1,000,000
Number of evaluation	3	5	1
Elite fraction	0.2	0.3	0.1
Imitation period	2	10	10

5.2. Comparison

The results in Table 4 and Fig. 6 show the final performance and learning curves of the four algorithms on Mujoco-based continuous control benchmarks. Each algorithm is trained for 5 M steps in Walker2d-v2, 2.5 M steps in Hopper-v2, and 1 M steps in HalfCheetah-v2 and Swimmer-v2. Table 4 presents the maximum (Max) reward obtained by each algorithm in the whole learning process, as well as the average (Mean), median (Median) and standard deviation (Std.) of 5 test results of each algorithm under different random seeds. The best statistical results have been bolded in each task. Fig. 6 presents the average proformance and error bars of 5 test results, which have been smoothed using a window of size 10.

Table 4 shows that the average performance of RIM exceeds the previous methods in all environments, and the maximum rewards obtained in the whole learning process and median performance exceed the previous methods in most environments. Both RIM and ERL have higher variances on different random seeds because the components, such as EA and DDPG, have high variances. However, the standard deviation of RIM performance is much lower than ERL in Hopper-v2 and Swimmer-v2.

The experimental results in Fig. 6 show that RIM can learn better than the previous algorithms in four tasks. In Walker2d-v2 and Hopper-v2 environments, the superiority of RIM is more significant, because the traditional off-policy reinforcement

Table 4
Final performance of EA, DDPG, ERL and RIM on 4 Mujoco-based continuous control benchmarks.

		Walker2d-v2	Hopper-v2	Halfcheetah-v2	Swimmer-v2
EA	Max	1339.11	1051.19	1755.27	**356.01**
	Mean	1143.68	1032.92	920.60	239.04
	Median	1073.38	1027.82	864.62	**301.93**
	Std.	**11.37%**	**0.08%**	65.49%	43.21%
DDPG	Max	3184.08	**3575.19**	6012.01	55.32
	Mean	543.94	484.29	2601.53	26.78
	Median	492.85	590.66	2796.43	29.25
	Std.	39.77%	67.12%	60.66%	27.86%
ERL	Max	4472.21	2392.49	5597.36	332.69
	Mean	1375.24	1780.90	5098.46	231.38
	Median	1476.77	1824.88	5064.25	222.56
	Std.	23.51%	30.23%	**5.54%**	21.37%
RIM(ours)	Max	**5532.66**	3439.74	**6151.81**	343.62
	Mean	**2852.08**	**2385.05**	**5399.68**	**297.43**
	Median	**3242.61**	**2386.84**	**5367.22**	287.49
	Std.	26.66%	19.66%	6.94%	**9.78%**

learning method cannot explore efficiently in such environments. EA has a stagnation period in the evolution process. Off-policy reinforcement learning individuals in ERL can help the evolutionary population to break through the stagnation period, but this often requires superior individuals in the population to generate a large amount of experiences. The recruitment mechanism of RIM enables outstanding individuals of EA to directly guide reinforcement learning individuals, instead of guiding them by generating experiences. Therefore, RIM can break through the stagnation period earlier than ERL, which allows RIM to learn faster.

We reimplemented the EA, DDPG and ERL algorithms. The ERL results we presented in Walker2d and Hopper environments are consistent with Khadka et al. [15]. The results in Halfcheetah are lower than Khadka et al., but close to those reported by Pourchot et al. [16]. In addition, we found that in Swimmer, the performance of EA, ERL and RIM is unstable. In the best case, they can reach 330 or more, but in the worst case, they perform less than 250, even less than 150 in EA.

It can also be inferred from Fig. 6 how hyperparameters of the RL part and the EA part affect the overall performance of RIM. In Swimmer-v2, DDPG is hard to learn. It can be inferred that the performance of RIM in this environment is mainly provided by EA. So the hyperparameter adjustment of the RL part might have little effect on RIM in Swimmer. But in other environments, the good performance of RIM is the joint result of the policy from RL and EA, because either algorithm alone cannot achieve the performance of RIM. Therefore, in these environments, the hyperparameters of the RL and EA parts affect the performance of RIM together.

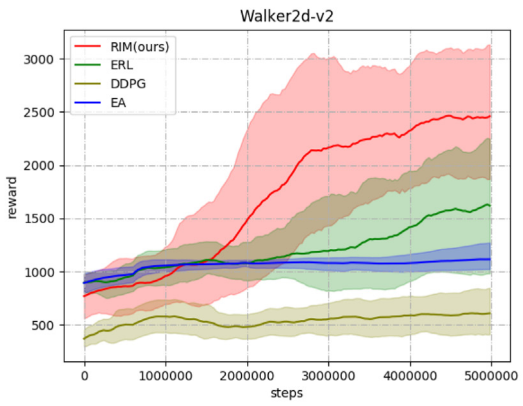
The performance of RIM heavily relies on the policies provided by the RL part. And because the RL method is considered to be sensitive to hyperparameters, RIM might also be sensitive to its hyperparameters, especially the hyperparameters of RL part. However, because EA can be seen as an extension to RL in RIM, even if RL learns a bad or sub-optimal policy due to hyperparameters, EA can still continue to optimize the policy provided by RL in the parameter space. Therefore, we believe that RIM is insensitive to hyperparameters compared with RL methods.

5.3. Component performance

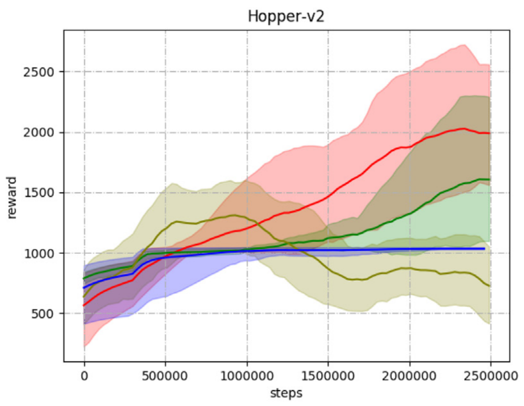
Fig. 7 shows the performance of three components in Walker2d and Hopper environments: RL components of RIM, imitation learning (IL) actors in RIM, and RL agents in ERL. The curves in Fig. 7 come from statistical results of 5 tests for the actors. The performance of RL agents in RIM is better than RL agents in ERL, which means that the recruitment mechanism can accelerate the learning process of RL actors. The performance of individuals trained by imitation learning in RIM improves as the performance of the RL components improves, which illustrates the effectiveness of off-policy imitation learning.

The externally injected individuals accepted by RIM's population are imitation learning individuals, while the externally injected individuals accepted by the ERL's population are RL actors. In Fig. 7, the performance of imitation learner in RIM outperforms RL agent in ERL, suggesting that externally injected individuals of the population of RIM are better. This can directly explain why the learning speed of RIM is faster than that of ERL.

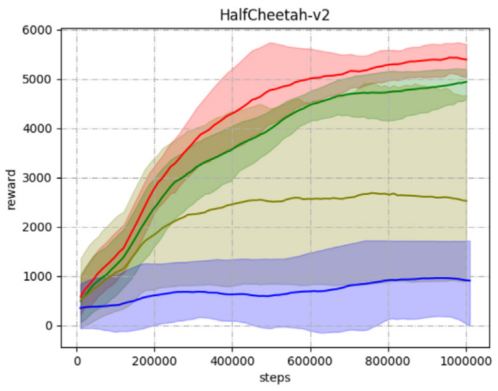
Table 5 presents the selection rate of actors trained through imitation learning during evolution. When the population selects excellent individuals, π_{wt} with higher performance has higher probability to be selected. On the contrary, π_{wt} with lower performance will have a higher probability to be discarded. The RL components perform better in Walker2d and Halfcheetah, so the selected probability of well-learned π_{wt} is higher. In Hopper, the RL components perform poorly before breaking through the long stagnation period. In Swimmer, the RL components cannot play a critical role in the whole learning process, which can be seen from the curve of DDPG in Fig. 6. These reasons cause π_{wt} cannot imitate a good policy, so the



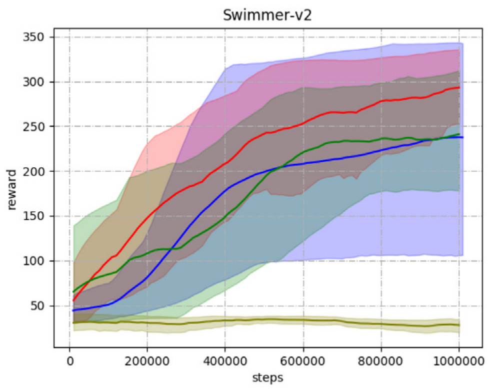
(a) Walker2d-v2



(b) Hopper-v2



(c) HalfCheetah-v2



(d) Swimmer-v2

Fig. 6. Learning curves of EA, DDPG, ERL and RIM on 4 Mujoco-based continuous control benchmarks.

probability that π_{wt} is selected is lower. On the whole, individuals performing imitation learning can be selected in each environment, indicating that the policy learned by imitation learning can accelerate the evolution of the population.

5.4. Soft update in recruitment

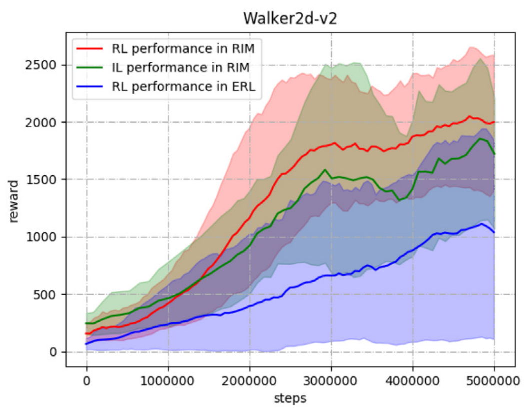
We tested the performance of RIM using hard update, and compared it with the original RIM. Fig. 8 shows the performance of RIM using hard update recruitment and soft update recruitment over 5 tests. As shown in Fig. 8, RIM using soft update recruitment performs slightly better than RIM using hard update recruitment.

Because in the process of guiding the gradient policy network learning, the soft update recruitment policy network can provide more stable $y^{(i)}$ when calculating Eq. (4), which ensures better consistency for learning process. Recruitment using hard update does not make the policy gradient network unable to learn. This is because as the population iterates, the changes of parameters between generations are usually small. Therefore, parameter changes of the recruitment policy network are small, which can also bring a certain degree of consistency to the learning process.

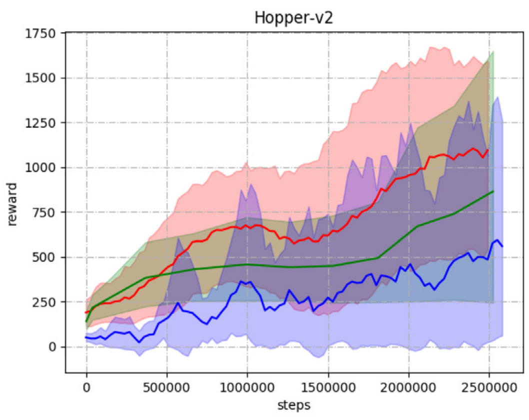
5.5. Ablation experiments

In order to further prove the effectiveness of the RIM components, we performed ablation experiments on RIM. We tested the performance of RIM without off-policy imitation learning (RIM-IL), RIM without using the recruitment network to estimate Q' (RIM-EA), and RIM without using the gradient policy network to estimate Q' (RIM-PG). We compared the test performance of these three RIM variants with RIM as well as ERL. The comparison results are averaged over 5 tests and are presented in Table 6 and Fig. 9.

According to the results in Table 6 and Fig. 9, RIM outperforms RIM-IL, which shows that imitation learning works. We believe that imitation learning can inject individuals with dual-policy RL agent's behavior patterns into the population, thereby accelerating the evolution of the population. Empirically speaking, the hyperparameters of the imitation learning



(a) Walker2d-v2



(b) Hopper-v2

Fig. 7. Performance of components of different algorithms.

Table 5
Selection rate for π_{wt} after training.

	Selected	Discarded
Walker2d-v2	59.87%	40.13%
Hopper-v2	30.92%	69.08%
HalfCheetah-v2	70.37%	29.63%
Swimmer-v2	24.44%	75.56%

part also affect the performance of RIM by affecting the imitation process. Bad hyperparameter settings of IL part will cause the performance of RIM to deteriorate, but even if the hyperparameter settings make the IL part completely invalid, the performance of RIM will not be significantly worse than that of RIM-IL. Performance of RIM is better than that of RIM-EA and RIM-PG, which shows that the dual-policy learning mode can learn better policy. This stems from the fact that the dual-policy can better estimate the Q' value, as revealed in Theorem 1.

To further illustrate the effectiveness of the dual-policy learning method in evolutionary reinforcement learning algorithms, we compared performance curves over 5 test results of RL components in RIM, RIM-EA, and RIM-PG algorithms. These curves are shown in Fig. 10. RIM's RL policy is better than RIM-EA, which indicates that RIM estimates the Q' value better than DDPG. RIM-PG's RL components perform poorly. This results from the fact that the Q' estimated by the recruitment network is not accurate in Walker2d-v2 and Hopper-v2.

5.6. Time

The test time during which the actor makes decisions depends on the speed that its network performs forward propagation. In experiments of this paper, actors of all algorithms have the same network structure, so the test time of all algorithms is the same in the same environment.

In the training process, the time consumption of EA is the least. DDPG needs to calculate the gradient and perform backpropagation so that it consumes much more time than EA. In addition to calculating gradient and performing backpropagation, ERL also needs to perform the evolution process, so it takes slightly more time than DDPG. The training time for performing the same steps in RIM is about 1.8 times than that in ERL. On the one hand, this is because RIM needs to perform additional imitation learning, and on the other hand, when calculating Eq. (3) and Eq. (4), RIM needs perform additional forward propagation process, such as the calculations of $Q(s_t, \pi_{ea}(s_t|\theta^{ea})|\theta^Q)$ and $Q'(s_{t+1}^{(i)}, \pi_{rl}(s_{t+1}^{(i)})|\theta^{Q'})$. Nonetheless, these additional calculations bring better progressive performance to RIM, so these additional costs are worth it. Fig. 11 shows the changes in the statistical training performance of RIM and ERL over time in the training process. The statistical training performance comes from 5 test results. Before 3 h, RIM and ERL have similar performance, but after training 3 h, the performance of RIM surpasses that of ERL.

6. Conclusion and future work

In this paper, we develop a dual-actors and single critic RL agent, which can be well combined with evolutionary algorithms. Based on this, we propose RIM for evolutionary reinforcement learning. This method not only outperforms the pre-

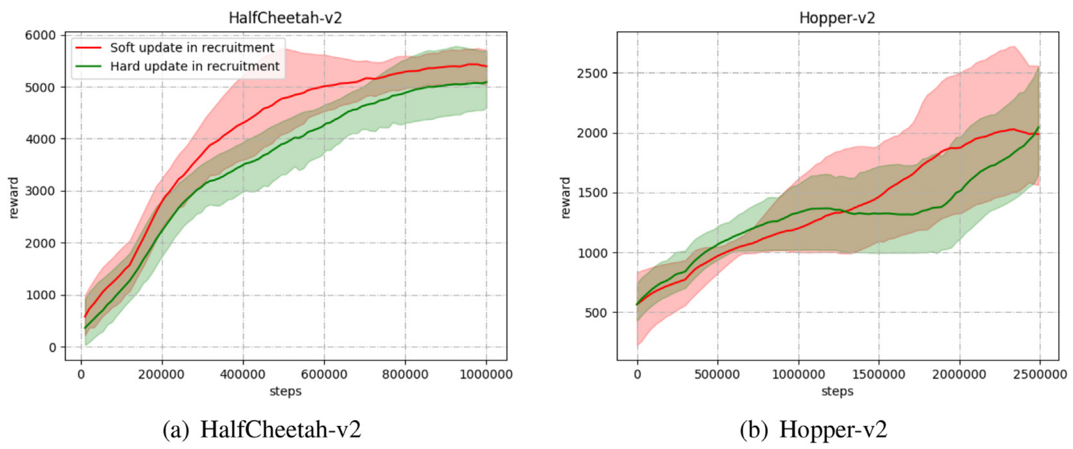


Fig. 8. Comparison curves between soft update recruitment and hard update recruitment.

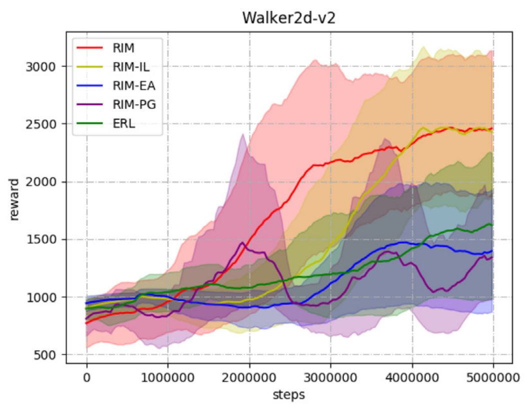
Table 6
Final performance of ERL, RIM and variants of RIM on 4 Mujoco-based continuous control benchmarks.

		Walker2d-v2	Hopper-v2	Halfcheetah-v2	Swimmer-v2
RIM	Mean	**2852.08**	**2385.05**	**5399.68**	**297.43**
	Median	**3242.61**	**2386.84**	**5367.22**	**287.49**
	Std.	26.66%	19.66%	6.94%	9.78%
RIM-IL	Mean	2075.33	1574.41	5229.43	269.68
	Median	1937.64	1095.01	5253.77	267.89
	Std.	26.43%	56.14%	7.5%	**9.45%**
RIM-EA	Mean	1948.16	1753.66	5001.21	258.96
	Median	2018.73	1749.26	4923.92	261.63
	Std.	49.10%	29.14%	7.67%	20.63%
RIM-PG	Mean	1777.30	1141.54	5142.67	222.60
	Median	1208.66	1078.56	5197.68	209.84
	Std.	54.97%	**15.23%**	13.96%	32.61%
ERL	Mean	1375.24	1780.90	5098.46	231.38
	Median	1476.77	1824.88	5064.25	222.56
	Std.	**23.51%**	30.23%	**5.54%**	21.37%

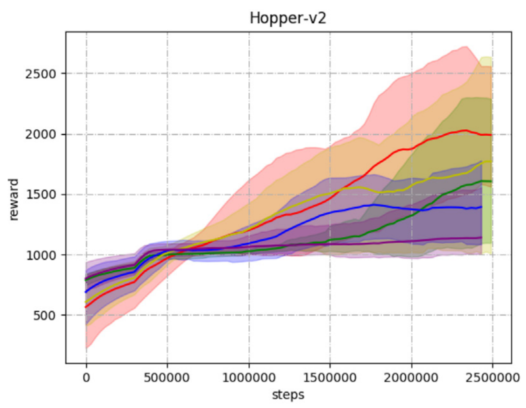
vious evolutionary and off-policy reinforcement learning algorithms, but also exceeds ERL both in overall performance and component performance. Finally, we experimentally demonstrate that the recruitment using soft update enables RL agent to learn faster than that using hard update.

Compared with the previous algorithms, RIM has comparative advantages in many aspects:

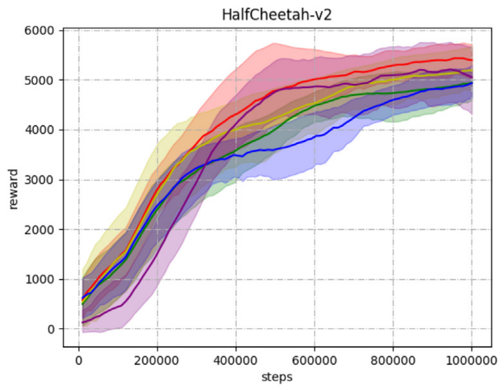
- Compared with a separate RL algorithm, RIM can use EA and the dual-policy structure to assist the exploration. And the dual-policy agent enables the RL agent based on DDPG to make better decisions and provide better target values even in the case of insufficient policy learning. Therefore, RIM can effectively solve the problems of insufficient RL exploration and inaccurate estimation of the target value in DDPG.
- Compared with a separate EA, RIM uses RL and IL to inject better individuals to the population to accelerate evolution. RIM can effectively solve the problem of low efficiency in EA.
- Compared with a separate IL algorithm, RIM does not require external expert data. RIM just uses IL to optimize the behavior of its internal components.
- Compared with ERL, RIM has improved the learning process of the RL agent in ERL. After integrating the information from EA, it can obtain a more accurate target value to guide the update of the critic. In addition, the imitation learning process directly improves the performance of individuals in the population. At the same time, this process strengthens the exploration of the parameter space in evolution. RIM can be regarded as optimization and expansion to ERL. RIM has higher sample efficiency and better asymptotic performance than ERL.



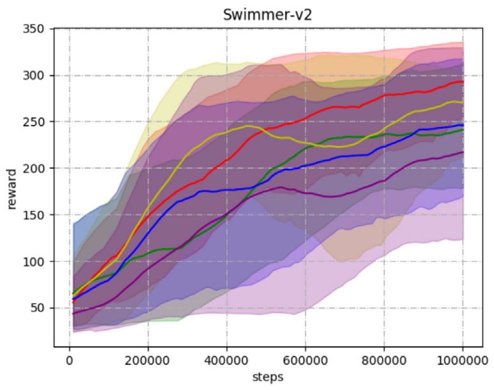
(a) Walker2d-v2



(b) Hopper-v2

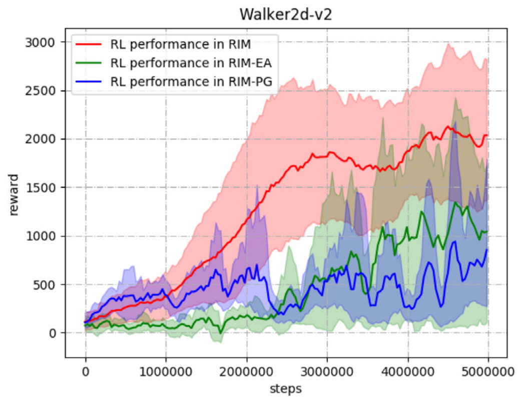


(c) HalfCheetah-v2

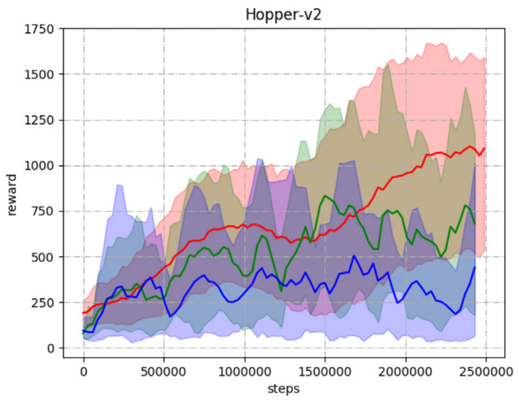


(d) Swimmer-v2

Fig. 9. Learning curves of ERL, RIM and variants of RIM on 4 Mujoco-based continuous control benchmarks.



(a) Wakler2d-v2



(b) Hopper-v2

Fig. 10. Learning curves of RL components in RIM, RIM-PG, and RIM-EA.

Future work can be explored from the following three aspects:

- RIM uses a standard evolutionary algorithm to iterate populations. Using other more efficient evolutionary algorithms may provide immediate improvements to RIM's performance, such as Natural Evolution Strategy (NES) [41], NeuroEvolution of Augmenting Topologies (NEAT) [42] and Transformation of Solutions Location (TSL) [43].

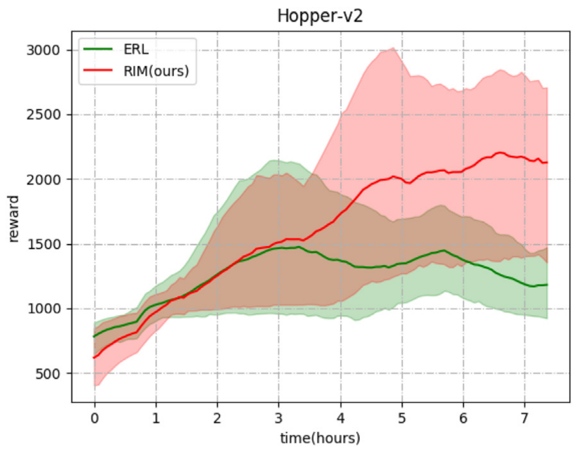


Fig. 11. Comparison of training time between ERL and RIM in Hopper-v2.

- The dual-actors and single critic RL agent in RIM is extended based on DDPG, but in fact, any off-policy actor-critic deep reinforcement learning methods can integrate this structure, such as Soft Actor-Critic (SAC) [44] and Twin Delayed Deep Deterministic policy gradient algorithm (TD3) [26]. Since these methods are superior to DDPG in most cases, integrating these methods to the dual-actors and single critic RL agent may improve RIM's performance.
- In RIM's experimental settings, at each time 100,000 reinforcement learning gradient updates are performed, then 30,000 imitation learning gradient updates are required. Parallel methods can be employed to reduce time for training algorithms.

CRediT authorship contribution statement

Shuai Lü: Conceptualization, Writing - review & editing, Supervision, Funding acquisition. **Shuai Han:** Conceptualization, Methodology, Software, Formal analysis, Validation, Writing - original draft. **Wenbo Zhou:** Writing - review & editing, Visualization. **Junwei Zhang:** Validation, Writing - review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We sincerely thank the anonymous reviewers for their careful work and thoughtful suggestions, which have greatly improved this article. This work was supported by the Natural Science Research Foundation of Jilin Province of China under Grant No. 20180101053JC; the National Key R&D Program of China under Grant No. 2017YFB1003103; and the National Natural Science Foundation of China under Grant No. 61300049.

References

[1] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, in: 4th International Conference on Learning Representations, 2016.
[2] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in: International Conference on Machine Learning, 2016, pp. 1928–1937..
[3] J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz, Trust region policy optimization, in: International Conference on Machine Learning, 2015, pp. 1889–1897..
[4] J. Li, L. Yao, X. Xu, B. Cheng, J. Ren, Deep reinforcement learning for pedestrian collision avoidance and human-machine cooperative driving, Inf. Sci. 532 (2020) 110–124.
[5] N. Pröllochs, S. Feuerriegel, B. Lutz, D. Neumann, Negation scope detection for sentiment analysis: A reinforcement learning framework for replicating human interpretations, Inf. Sci. 536 (2020) 205–221.
[6] H. Wang, X. Hu, Q. Yu, M. Gu, W. Zhao, J. Yan, T. Hong, Integrating reinforcement learning and skyline computing for adaptive service composition, Inf. Sci. 519 (2020) 141–160.
[7] T. Salimans, J. Ho, X. Chen, S. Sidor, I. Sutskever, Evolution strategies as a scalable alternative to reinforcement learning, arXiv preprint arXiv:1703.03864..
[8] K. Zhang, C. Shen, J. He, G.G. Yen, Knee based multimodal multi-objective evolutionary algorithm for decision making, Inf. Sci. 544 (2021) 39–55.
[9] A. Giusti, J. Guzzi, D.C. Ciresan, F. He, J.P. Rodriguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G.D. Caro, D. Scaramuzza, L.M. Gambardella, A machine learning approach to visual perception of forest trails for mobile robots, IEEE Robotics Automation Lett. 1 (2) (2016) 661–667.

[10] F. Codevilla, M. Miiller, A. López, V. Koltun, A. Dosovitskiy, End-to-end driving via conditional imitation learning, in: 2008 IEEE International Conference on Robotics and Automation (ICRA), 2018, pp. 1–9.
[11] S. Zhang, O.R. Zaiane, Comparing deep reinforcement learning and evolutionary methods in continuous control, arXiv preprint arXiv:1712.00006..
[12] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M.A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533.
[13] H. Wang, Y. Wu, G. Min, J. Xu, P. Tang, Data-driven dynamic resource scheduling for network slicing: A deep reinforcement learning approach, Inf. Sci. 498 (2019) 106–116.
[14] J. Xu, L. Yao, L. Li, M. Ji, G. Tang, Argumentation based reinforcement learning for meta-knowledge extraction, Inf. Sci. 506 (2020) 258–272.
[15] S. Khadka, K. Tumer, Evolution-guided policy gradient in reinforcement learning, Adv. Neural Inform. Process. Syst. (2018) 1196–1208.
[16] A. Pourchot, O. Sigaud, CEM-RL Combining evolutionary and gradient-based methods for policy search, in: 7th International Conference on Learning Representations, 2019.
[17] S. Ross, J.A. Bagnell, Reinforcement and imitation learning via interactive no-regret learning, arXiv preprint arXiv:1406.5979..
[18] E. Uchibe, Cooperative and competitive reinforcement and imitation learning for a mixture of heterogeneous learning modules, Front. Neurorobotics 12 (2018) 61.
[19] D.V. Vargas, Evolutionary reinforcement learning: general models and adaptation., in, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, 2018, pp. 1017–1038.
[20] M.M. Drugan, Reinforcement learning versus evolutionary computation: A survey on hybrid algorithms, Swarm Evol. Comput. 44 (2019) 228–246.
[21] F. Zou, G.G. Yen, L. Tang, C. Wang, A reinforcement learning approach for dynamic multi-objective optimization, Inf. Sci. 546 (2021) 815–834.
[22] H. Tan, K. Balajee, D. Lynn, Integration of evolutionary computing and reinforcement learning for robotic imitation learning, in: 2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2014, pp. 407–412..
[23] S. Whiteson, P. Stone, Evolutionary function approximation for reinforcement learning, J. Mach. Learn. Res. 7 (5) (2006) 877–917.
[24] J. Kober, J. Peters, Imitation and reinforcement learning, IEEE Robotics Autom. Magazine 17 (2) (2010) 55–62.
[25] S. Khadka, S. Majumdar, S. Miret, E. Tumer, T. Nassar, Z. Dwiel, Y. Liu, K. Tumer, Collaborative evolutionary reinforcement learning, in: International Conference on Machine Learning, 2019, pp. 3341–3350..
[26] S. Fujimoto, H. van Hoof, D. Meger, Addressing function approximation error in actor-critic methods, in: International Conference on Machine Learning, 2018, pp. 1582–1591..
[27] G. Li, Q. Lin, W. Gao, Multifactorial optimization via explicit multipopulation evolutionary framework, Inf. Sci. 512 (2020) 1555–1570.
[28] J. Zhang, J.X. Huang, Q.V. Hu, Boosting evolutionary optimization via fuzzy-classification-assisted selection, Inf. Sci. 519 (2020) 423–438.
[29] S. Ross, G.J. Gordon, J.A. Bagnell, No-regret reductions for imitation learning and structured prediction, AISTATS, 2011.
[30] A. Attia, S. Dayan, Global overview of imitation learning, arXiv preprint arXiv:1801.06503..
[31] H.V. Hasselt, Double Q-learning, Advances in Neural Information Processing Systems, 2010, pp. 2613–2621.
[32] H. van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double Q-learning, in: Thirtieth AAAI conference on Artificial Intelligence, 2016, pp. 2094–2100..
[33] E. Todorov, T. Erez, Y. Tassa, Mujoco: A physics engine for model-based control, 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2012, pp. 5026–5033.
[34] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, OpenAI gym, arXiv preprint arXiv:1606.01540..
[35] Y. Duan, X. Chen, R. Houthooft, J. Schulman, P. Abbeel, Benchmarking deep reinforcement learning for continuous control, in: International Conference on Machine Learning, 2016, pp. 1329–1338..
[36] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, D. Meger, Deep reinforcement learning that matters, in: Thirty-Second AAAI Conference on Artificial Intelligence, 2018, pp. 3207–3214..
[37] R. Islam, P. Henderson, M. Gomrokchi, D. Precup, Reproducibility of benchmarked deep reinforcement learning tasks for continuous control, arXiv preprint arXiv:1708.04133..
[38] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, Pytorch: An imperative style, high-performance deep learning library, Adv. Neural Inform. Process. Syst. (2019) 8026–8037.
[39] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980..
[40] G.E. Uhlenbeck, L.S. Ornstein, On the theory of the brownian motion, Phys. Rev. 36 (1930) 823.
[41] D. Wierstra, T. Schaul, J. Peters, J. Schmidhuber, Natural evolution strategies, 2008 IEEE Congress on Evolutionary Computation, 2008, pp. 3381–3387.
[42] K.O. Stanley, R. Miikkulainen, Evolving neural networks through augmenting topologies, Evol. Comput. 10 (2) (2002) 99–127.
[43] Z. Liang, W. Hou, X. Huang, Z. Zhu, Two new reference vector adaptation strategies for many-objective evolutionary algorithms, Inf. Sci. 483 (2019) 332–349.
[44] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine, Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, in: International Conference on Machine Learning, 2018, pp. 1856–1865..