Adaptive Evolutionary Reinforcement Learning Algorithm with Early Termination Strategy

Xiaoqiang Wu
College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China
wuxiaoqiang2021@email.szu.edu.cn

Qingling Zhu
National Engineering Laboratory for Big Data System Computing Technology, Shenzhen University, Shenzhen, China
zhuqingling@szu.edu.cn

Qiuzhen Lin*
College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China
qiuzhlin@szu.edu.cn

Weineng Chen
School of Computer Science and Engineering, South China University of Technology, Guangzhou, China
cwnraul634@aliyun.com

Jianqiang Li
National Engineering Laboratory for Big Data System Computing Technology, Shenzhen University, Shenzhen, China
lijq@szu.edu.cn

ABSTRACT

Evolutionary reinforcement learning algorithms (ERLs), which combine evolutionary algorithms (EAs) with reinforcement learning (RL), have demonstrated significant success in enhancing RL performance. However, most ERLs rely heavily on Gaussian mutation operators to generate new individuals. When the standard deviation is too large or small, this approach will result in the production of poor or highly similar offspring. Such outcomes can be detrimental to the learning process of the RL agent, as too many poor or similar experiences are generated by these individuals. In order to alleviate these issues, this paper proposes an Adaptive Evolutionary Reinforcement Learning (AERL) method that adaptively adjusts both the standard deviation and the evaluation process. By tracking the performance of new individuals, AERL maintains the mutation strength within a suitable range without the need for additional gradient computations. Moreover, the proposed AERL approach early terminates unnecessary evaluations and discards experiences arising from poor individuals, thereby resulting in enhanced learning efficiency. Empirical assessments conducted on a variety of continuous control problems demonstrate the effectiveness of the AERL method.

KEYWORDS

Reinforcement Learning; Evolutionary Algorithm; Evolutionary Reinforcement Learning

*Corresponding author.

Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – 10, 2024, Auckland, New Zealand. © 2024 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

ACM Reference Format:
Xiaoqiang Wu, Qingling Zhu, Qiuzhen Lin, Weineng Chen, and Jianqiang Li. 2024. Adaptive Evolutionary Reinforcement Learning Algorithm with Early Termination Strategy. In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024*, IFAAMAS, 9 pages.

1 INTRODUCTION

Reinforcement Learning (RL) has been widely applied in various areas , showcasing capabilities that even surpass human performance [2, 17, 29], primarily owing to the successful integration of RL with deep neural networks [16]. Nonetheless, numerous common challenges persist in RL algorithms, such as the difficulty in deriving informative learning signals from sparse rewards and the limitation in promoting diverse exploration [13].

Over the past few years, several research efforts have attempted to introduce Evolutionary Algorithms (EAs) to help address these issues. Notably, in [22, 25], EAs have been explored as a competitive alternative to Reinforcement Learning (RL). These methods adopt the policy networks and their cumulative rewards as individuals and fitness values, respectively. While this approach can naturally overcome the influence from sparse rewards, it generally suffers from lower sample efficiency than RL methods because of the neglect of the specific information in each step. Except for replacing RL algorithms with EAs, a more promising scheme is to combine both of them and leverage their advantages from two domains. Among these hybrid methods that integrate EA and RL, Evolutionary Reinforcement Learning (ERL) [13] is one of the most popular algorithms. ERL maintains an actor population optimized by Genetic Algorithm (GA) and an additional actor optimized by RL algorithm. The actor population can explore different regions of the solution space and provide diverse experiences for the RL actor's training. Additionally, the updated RL actor is copied into the population and used to produce the next generation. By combining GA and RL in a suitable way, ERL can combine the strengths

of both methods and demonstrate superior performance compared to the original RL algorithm or GA.

While ERL [13] has exhibited the potential and superiority of hybrid methods, it employs completely random crossover and mutation operators that could potentially harm the functionality of neural networks. To address this weakness, Proximal Distilled Evolutionary Reinforcement Learning (PDERL) [3] devises two genetic operators based on backpropagation that are more stable and efficient than the operators used in the original ERL. Incorporating these operators with ERL, PDERL significantly improves its performance. Nevertheless, both operators in PDERL rely on gradient computation, which increases its computational burden. Furthermore, ERL and PDERL collect the entire population's experiences irrespective of the quality of single individual. In reality, newly generated individuals may be poor due to the randomness of mutation, and their experiences may not benefit the learning of RL agents. To alleviate the computational burden, we propose an adaptive mutation operator that adjusts the mutation strength within a proper range based on the performance of the newly generated individuals. Additionally, our method employs an early termination strategy that determines the termination of the evaluation process. Specifically, the evaluation of poor individuals is terminated early, and their experiences are discarded. The main contributions of this article are as follows:

1) An adaptive mutation operator is proposed for evolutionary reinforcement learning algorithms (ERLs), which can effectively control the strength of mutation and improve practical performance. This operator does not require extra gradient computation and has a low computational cost.
2) An early termination strategy is designed to further enhance the efficiency of ERLs. This strategy discards useless evaluations and experiences from poor-performing individuals, leading to improved learning efficiency and algorithm performance.
3) The above approaches are integrated with RL algorithms to form an Adaptive Evolutionary Reinforcement Learning called AERL, which outperforms other compared algorithms in various continuous control problems.

The remainder of this paper is organized as follows. Section 2 provides background information and related work on ERLs. Section 3 details the proposed adaptive mutation operator, evaluation strategy, and their integration with RL algorithms. Section 4 compares AERL to state-of-the-art ERLs and RL algorithms, and includes ablation experiments to demonstrate the advantages of our method. Finally, Section 5 provides conclusions and future work.

2 BACKGROUND AND MOTIVATION

This section provides some related background information. First, a brief introduction to GAs is presented, which are commonly used as the EA component in many ERLs. Next, a brief overview of the most frequently used RL algorithms in ERLs is provided. Finally, we describe some representative ERLs.

2.1 GA

GAs [31, 34] are population-based heuristic search algorithms that mimic natural selection and biological evolution. GA's reproduction process includes three genetic operators: selection, crossover (or recombination), and mutation [6]. Similar to natural selection, the selection operator favors individuals with higher fitness values [1]. There are typically stochastic operations in the selection operators, such as tournament selection [15] and roulette wheel selection [32], which introduce diversity into the population. In nature, the chromosomes of parents are combined to produce the next generation. The crossover operators in GA simulate this process. For instance, single-point crossover [8] swaps the chromosomes of parents before a random point. The mutation operator modifies the chromosomes of individuals with a certain probability. When the chromosomes represent the weights of a neural network, Gaussian mutation [8] is widely used as the base of the mutation operator.

2.2 DDPG and TD3

Deep Deterministic Policy Gradient (DDPG) [14] extends the idea of DQN [16] to continuous action domains, and is also based on the Deterministic Policy Gradient (DPG) algorithm [24]. To produce continuous actions, DDPG employs an actor-network μ to represent the policy and a critic network Q to approximate the action-value function, which forms the actor-critic architecture [27]. Additionally, DDPG also uses target networks μ' and Q'. At each time step, the agent receives a state s_t and selects an action a_t using the policy network. Then, the environment returns a reward r_t and the next state s_{t+1} to the agent. After the interaction between the agent and environment, the tuple (s_t, a_t, r_t, s_{t+1}) is stored in the experience replay buffer R. To optimize the actor and critic networks, losses are computed using a batch of tuples. Specifically, the critic network is updated by minimizing the loss L as follows:

$$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2 \tag{1}$$

where $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$, and θ means network's parameter. The actor is updated by the sampled policy gradient:

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

Twin delayed deep deterministic policy gradient (TD3) [9] indicates that DDPG overestimates the value of action and utilizes two value function networks to alleviate this problem, which is similar to Double Q-learning [12]. In TD3, two critic networks provide separate estimates of the value function, and the minimum estimate is used. The value target y_i in the critic loss is calculated using Eq. (3):

$$y_i = r_i + \gamma\, min_{j=1,2}\, Q'_j(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'_j}) \tag{3}$$

In addition, TD3 updates the actor and target networks after updating the critic for multiple times to reduce estimation error.

2.3 ERLs and Motivation

To combine the benefits of EA and RL, various hybrid algorithms have been proposed. One such approach is ERL [13], which integrates GA and DDPG to optimize a population of actors and an extra RL actor, respectively. The actor population generates diverse experiences to train the RL actor, while the RL actor is periodically inserted into the population to provide new individuals. Although ERL has demonstrated its advantages on continuous control tasks, its genetic operators are damaging and may cause forgetting of the

learned information. To tackle this issue, Proximal Distilled Evolutionary Reinforcement Learning (PDERL) [3] proposes proximal mutation and distillation crossover. Proximal mutation utilizes the sum of gradients to reduce the mutation strength of each weight, increasing the stability of the mutation. Distillation crossover exploits parents' experiences to train their child using Imitation Learning [18]. These two operators outperform the original operators on the same tasks. Motivated by the framework of ERL, several studies explore schemes that leverage the advantages of more methods. Cross-Entropy Method Reinforcement Learning (CEM-RL) [19] proposes a different framework that integrates RL algorithms with CEM [11, 21]. CEM-RL applies RL gradient steps directly to half of the population and employs the top-performing half to generate individuals, so it can reduce the impact of gradient steps that decrease the performance. Competitive and Cooperative Heterogeneous Deep Reinforcement Learning (C2HRL) [33] uses a heterogeneous agent pool to force agents to compete for computation resources and cooperate with each other for more efficient exploration in the solution space. C2HRL's agent pool includes agents optimized by TD3, Soft Actor-Critic (SAC) [10], and EA. By leveraging multiple excellent optimization methods, C2HRL can gain more advantages from different areas than simple combination schemes. Evolution-based Soft Actor Critic (ESAC) [26] combines ES with SAC and utilizes hindsight crossover to facilitate skill transfer between individuals. Furthermore, ESAC maximizes the mutation rate of ES through its proposed automatic mutation tuning to enhance hyperparameter robustness.

Although many sophisticated combination schemes or operators have been proposed to improve the practical performance in different environments, they ignore the difference in experiences collected from ERLs and RL algorithms. In RL algorithms, new experiences are generally collected from the current policy, and even if action noise is used to improve exploration, it only slightly changes the agent's behavior. For ERLs, experiences come from the RL agent and the agent population. However, ERLs add random perturbations to the weights of the policy network to evolve the individuals, which can destroy the functionality of neural networks and produce poor individuals that are unhelpful for exploration. To prevent these individuals from influencing agents' learning, we identify them, terminate their evaluations during the evaluation process and discard their experiences. Besides, the strength of mutation affects the quality of newly produced individuals significantly. To maintain the mutation strength within a reasonable range without increasing massive extra computation, we vary the mutation strength based on the performance of new individuals. Moreover, by controlling the mutation strength, the ratio of new poor individuals can be reduced to some extent. The details and practical implementation of our method are discussed in the next section.

3 PROPOSED ALGORITHM

This section introduces the details of our proposed AERL algorithm. The general framework of AERL is illustrated in Fig. 1. The algorithm involves several main steps, including the evaluation of the RL actor π_{RL} and the population pop_π, the mutation of the population, and the gradient update of the RL agent. It is worth noting that the evaluation and update of the population will be executed

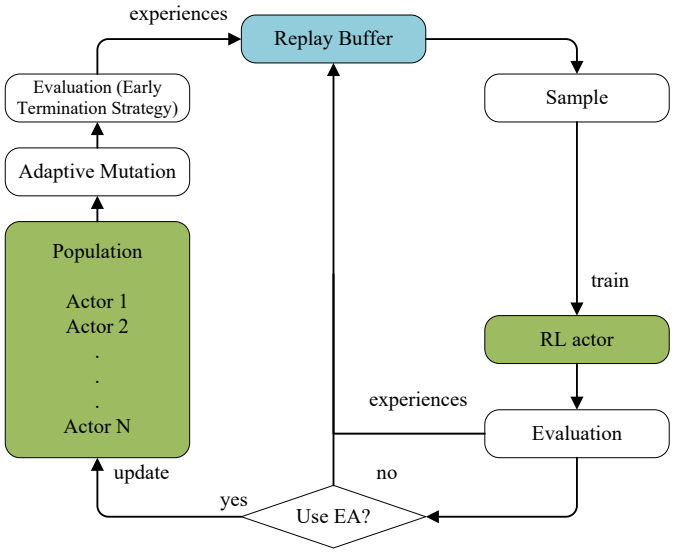


Figure 1: The algorithm framework of AERL

only if the RL agent makes no progress in the last K evaluations. This means that the evolutionary part of the algorithm (i.e., the left half in Fig. 1) will not be executed when the RL agent can make progress quickly. Finally, if the number of timesteps reaches the maximum value, the algorithm terminates and the best individual in the population is selected as the final solution.

3.1 Adaptive Mutation

When the weights of a neural network are the variables of mutation, the standard variation σ of the Gaussian mutation is critical. If σ is too large, it can disrupt the functionality of the neural network. Conversely, if σ is too small, the mutation operator will have almost no impact on the neural network, and the newly produced individuals will be similar to the old ones. Therefore, we propose an adaptive mutation operator that keeps the standard variation within an appropriate range. In PDERL [3], the strength of mutation of each weight is scaled by its gradient, but this approach requires extra gradient computation. Our method varies σ based on the performance of new individuals, which only requires very simple computation.

Algorithm 1 provides the pseudo-code for our mutation operator. This mutation operator requires a boolean variable, *increase*, as input. If this variable is true, the standard deviation σ will be increased in the next step, and vice versa. The value of *increase* is determined after evaluating the population. Specifically, *increase* will be set to false if all fitness values of the new individuals are smaller than half of the elitists' fitness; otherwise, it will be set to true. To make the change of σ more stable, we use two variables, u and l, which store the value of σ in the last decrease or increase, respectively. When we need to increase the value of σ, if σ is significantly less than u, the value of σ will be set to half of the sum of σ and u. Otherwise, the value of σ will be doubled. The steps for decreasing σ are similar to the above process. Finally, all new individuals are produced by mutating the elitists in the population.

Algorithm 1: Adaptive_Mutation

Input: population pop_π;
elitist e;
$increase$;
σ;
upper bound u;
lower bound l;
Output: pop_π, σ, u, l

```
if increase then
    l = σ;
    if σ ≤ u × 0.99 then
        σ = (σ + u)/2;
    else
        σ = σ × 2;
else
    u = σ;
    if σ × 0.99 > l then
        σ = (σ + l)/2;
    else
        σ = σ/2;
for π_i ∈ pop_π do
    if i ≠ e then
        θ_i ← θ_e + σ ∗ x, x ∼ N(0, I);
```

Algorithm 2: Evaluation with Early Termination Strategy

Input: actor π;
replay buffer R;
minimal timestep m;
average reward avg;
Output: $fitness$

```
Initialize fitness = 0 , empty buffer Q;
Reset environment and get initial state s_0;
while env is not done do
    Select action a_t = π(s_t|θ^π);
    Execute action a_t, get reward r_t and new state s_{t+1};
    fitness = fitness + r_t;
    Append transition (s_t, a_t, r_t, s_{t+1}) to Q;
    if t > m then
        if fitness < avg_t then
            break; // early termination
        if done then
            Append Q to R;
    update avg;
return fitness;
```

3.2 Early Termination Strategy

In this section, we propose a novel early termination strategy for evaluation operation. Due to the inherent randomness of the mutation operator, some new individuals in the population may have very poor performance, which can hinder exploration and provide no benefit to the learning of RL agent. Furthermore, evaluating these individuals can be a waste of time and resources, as the evaluation process often requires extensive interactions with the environment that may be costly or even hazardous in some real-world scenarios. To address this issue, we terminate the evaluation process of such individuals prematurely and discard their experiences.

Algorithm 2 provides the pseudo-code of our early termination strategy. The variable *average* stores the average cumulative reward of each step. The minimal timestep m is a hyperparameter, which is used to avoid the influence of randomness in the early interaction period. We compare the current cumulative reward to the average cumulative reward when the timestep t is larger than m. If the current value is smaller than the average value, we terminate the evaluation and discard the experiences in the previous steps. To store these transitions, we use a temporary replay buffer Q. If the interaction is done, the experiences in Q will be appended to R. Furthermore, the variable *average* is updated after every interaction.

Algorithm 3: AERL

```
Initialize actor π_RL, population pop_π, elitist e, increase, σ,
  upper bound u and lower bound l;
while True do
    f_RL = Evaluation(π_RL, R, m, avg) ;
    if RL agent makes no progress in last K evaluations then
        Adaptive_Mutation(pop_π, e, increase, σ, u, l) ;
        for actor π_i ∈ pop_π do
            f_i = Evaluation(π_i, R, m, avg); // Algorithm 2
        Update increase by f ;
        Select elitists e by f ;
    if f_RL > max(f) then
        Copy π_RL into the population ;
    Execute gradient update in RL algorithm ;
```

3.3 Complete Algorithm

We integrate our adaptive mutation operator and evaluation strategy with RL algorithms to create a novel algorithm called AERL. Algorithm 3 presents the pseudo-code of AERL. We utilize EA only when the RL agent makes no progress in the last K evaluations to reduce unnecessary evaluations for the population. If the RL agent achieves a higher score than the best individual in the population, it will be copied into the population.

The algorithm initializes the population and other variables in line 1 and then executes the while loop until the number of timesteps reaches the maximum value. At the start of each iteration, the RL actor is evaluated and a fitness score is returned. Next, if the RL actor fails to make progress in the last K evaluations, the EA part will be executed. Specifically, the population will be mutated using our adaptive mutation operator and evaluated. Additionally, the value of *increase* is updated as described in Section 3.1, and new elitists are selected based on the fitness values. If the fitness score of the RL actor is larger than the maximum fitness value in

the population, the algorithm copies the RL actor into the population. Finally, the algorithm executes the gradient update in the RL algorithm, and the next iteration of the while loop begins.

4 EXPERIMENTAL STUDIES

4.1 Test Problems

To evaluate the effectiveness of our method, we utilize the popular mujoco simulation environments [4, 28] widely used in RL fields [7, 14, 23]. Our evaluation focuses on five robot locomotion problems, namely HalfCheetah, Swimmer, Hopper, Ant, and Walker2d. The goal of these problems is to apply torque on the robots' joints to complete specific locomotion tasks. These environments have different features and can be used to test specific abilities of algorithms. For example, HalfCheetah is a basic environment for continuous control, while Ant has a high-dimensional state space that presents more challenges for learning agents. In Hopper and Walker2d, episodes are terminated if the agent's health status deteriorates, which may occur due to the selected terrible actions. This feature increases the difficulty of exploration and may cause the learning process to halt prematurely. In Swimmer, the reward for a single step can be deceptive for RL algorithms, leading to poor performance [3, 13].

Apart from evaluating our method on Mujoco environments, we also apply it to four classic control problems to test its performance more fully. These problems include BipedalWalker, LunarLanderContinuous, MountainCarContinuous, and Pendulum. Compared to the Mujoco environments, the four classic control problems present different challenges. In BipedalWalker and LunarLanderContinuous, the agent receives a significant negative reward from the environment when it falls or crashes, making exploration more difficult. Although the state and action dimensions of the MountainCarContinuous environment are small, the reward in this environment is sparse, with the agent only receiving a positive reward when it reaches the top of the hill. Until then, it merely receives negative rewards as a cost for taking actions, leaving the agent without an efficient guide.

4.2 Performance Metric

The performance of agents is measured by the average reward over five episodes without exploration noise. To ensure a fair comparison between RL algorithms and population-based algorithms, only the individual with the highest fitness within the population is tested, and the steps of each individual in the population are accumulated. In addition, same as ERL and PDERL, the reported results are the average values from five independent runs with different random seeds.

4.3 Experimental Setting

To assess the practical efficacy of the proposed algorithm, we conduct a comparative analysis against three competitive evolutionary reinforcement learning algorithms (ERL [13], PDERL [3], ESAC [26]), as well as two state-of-the-art RL algorithms (TD3 [9] and PPO [23]).

For a fair comparison with ERL and PDERL, our algorithm is implemented using DDPG. However, in the HalfCheeetah environment, the performance of AERL with DDPG does not outperform TD3. Therefore, to fully demonstrate the advantages of our method, we also implement AERL with TD3 and report the results for this environment. All compared algorithms are implemented using their published implementations. We set the hyperparameters of each algorithm to the values suggested in the corresponding papers or to default values in the authors' implementation. For the number of steps in experiments, we follow the setting in ERL, using 2 million steps for HalfCheetah and Swimmer, 4 million for Hopper, 6 million for Ant, and 8 million for Walker2d.

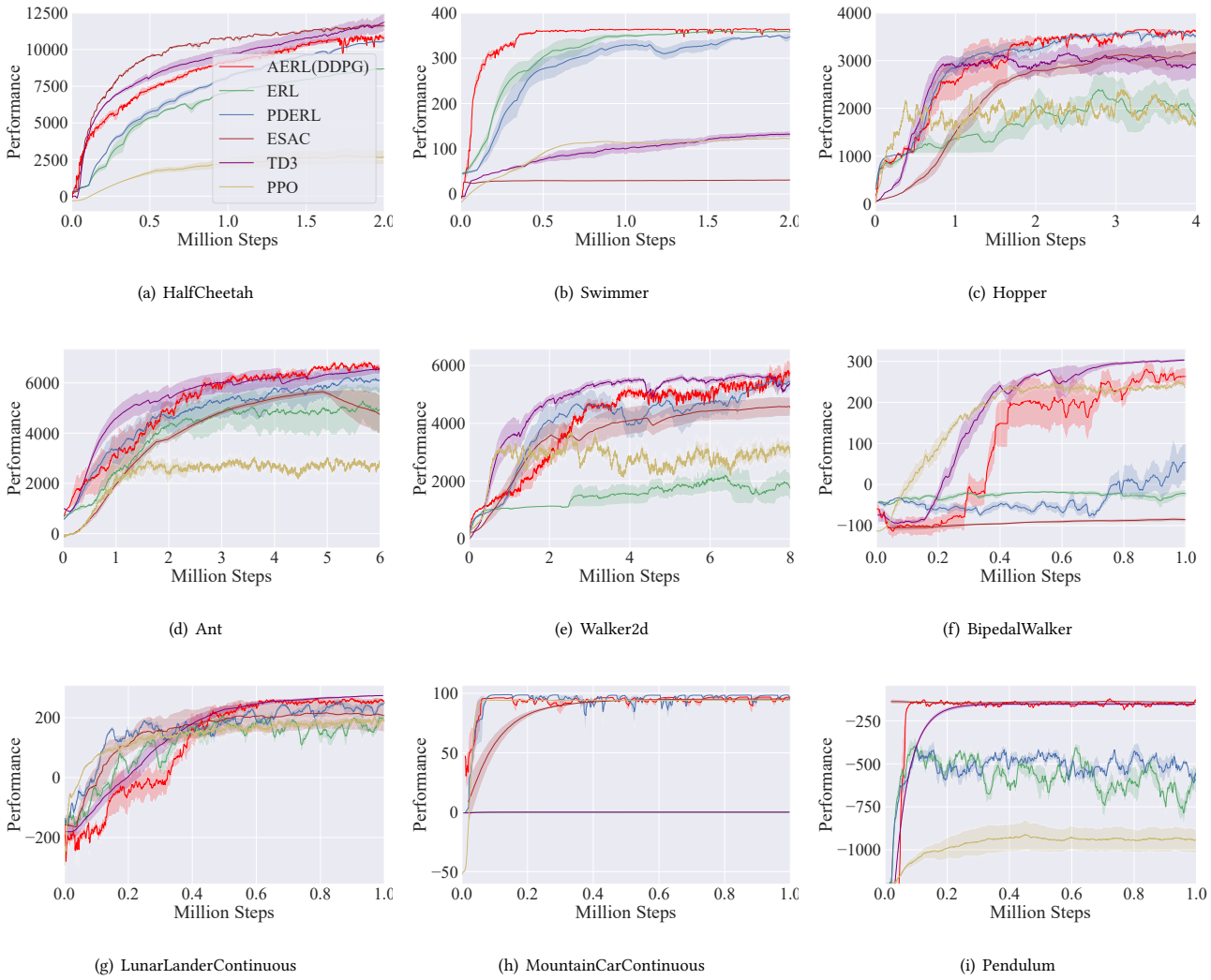
Most of the hyperparameters of our method are identical to those in ERL [13]. The specific hyperparameters used in our method are detailed as follows: we set K to 10, and we set the minimal timesteps m to 100 for Ant, LunarLanderContinuous and BipedalWalker; 200 for Swimmer; and 50 for other environments. For Hopper and Walker2d, we do not terminate the evaluation prematurely, as the interaction ends automatically when the agent becomes unhealthy. We also do not terminate the evaluation prematurely in MountainCarContinuous and Pendulum, because of the sparse reward and the short length of the episode, respectively.

4.4 Experimental Results

Figure 2 depicts the learning curves of the compared algorithms in solving five continuous control problems, with the shaded areas indicating standard variation of performance. To fully showcase the advantages of our method, we conduct additional experiments comparing AERL(TD3) with the original TD3, as demonstrated in Figure 3(a). Table 1 presents the mean and standard deviation of final performances in various mujoco test environments. On HalfCheetah, we report the result of AERL(DDPG) in this table, while the final result of AERL(TD3) is 12426 ± 131. AERL outperforms ERL, PDERL, ESAC, and PPO in all mujoco environments. AERL(DDPG) outperforms TD3 significantly on Swimmer, Hopper, and Walker2d and slightly on Ant. Although TD3 performs better than AERL with DDPG in the HalfCheetah environment, this limitation is due to the used RL algorithm, and we demonstrate that AERL(TD3) can outperform TD3 in this environment. PPO, on the other hand, does not perform well on these robot locomotion simulation problems, which may be attributed to its on-policy feature. Generally, on-policy RL algorithms are less sample efficient than off-policy RL algorithms [30]. Consequently, it may be challenging for on-policy RL algorithms to outperform off-policy RL algorithms when the number of timesteps is equal.

As HalfCheetah is a stable environment, an agent's performance changes slightly in different evaluation episodes, and the learning curves exhibit a steadily increasing trend. Therefore, TD3-based algorithms exhibit the best performance in this environment as TD3 has higher sample efficiency than DDPG. Our proposed method, AERL with TD3, outperforms all other compared algorithms in this environment. Although AERL with DDPG does not perform better than TD3, it still outperforms other DDPG-based algorithms.

The hybrid algorithms exhibit a clear advantage over RL algorithms on Swimmer, primarily due to the effectiveness of the evolutionary algorithms (EAs) they incorporate. While the Swimmer benchmark is challenging for RL algorithms to learn, it is relatively easier for EAs and standalone EA can also perform well on Swimmer [13]. This is likely because EAs rely only on the sum of

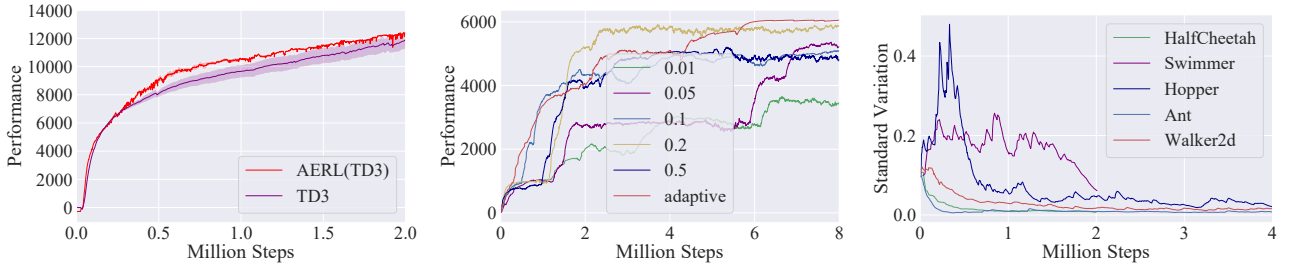


(a) HalfCheetah (b) Swimmer (c) Hopper

(d) Ant (e) Walker2d (f) BipedalWalker

(g) LunarLanderContinuous (h) MountainCarContinuous (i) Pendulum

Figure 2: Learning curves on mujoco continuous control benchmarks and classic control problems



(a) (b) (c)

Figure 3: (a) The performance comparison between AERL(TD3) and TD3 on HalfCheetah; (b) The performance comparison between AERL and its fixed standard variation variants; (c) The variation of σ in multiple test environments

Table 1: The mean and standard variation of final performance in mujoco test environments and classic control benchmarks

	Environment	AERL(DDPG)	ERL	PDERL	ESAC	TD3	PPO
Mujoco	HalfCheetah	10834 ± 818	8697 ± 108	10622 ± 221	11611 ± 336	11880 ± 1327	2660 ± 1251
	Swimmer	360 ± 8	358 ± 4	347 ± 13	31 ± 2	132 ± 13	122 ± 2
	Hopper	3643 ± 68	1850 ± 608	3546 ± 97	3163 ± 463	2900 ± 650	1656 ± 278
	Ant	6645 ± 329	4996 ± 1991	6116 ± 1075	4749 ± 1590	6552 ± 419	2872 ± 295
	Walker2d	5795 ± 891	1666 ± 835	5425 ± 696	4530 ± 635	5386 ± 227	2822 ± 646
Classical	BipedalWalker	264 ± 57	−25 ± 31	54 ± 107	−83 ± 13	304 ± 9	237 ± 15
	LunarLanderContinuous	253 ± 16	194 ± 34	248 ± 17	207 ± 107	273 ± 7	188 ± 24
	MountainCarContinuous	97 ± 1	0 ± 0	94 ± 1	95 ± 0	0 ± 0	94 ± 0
	Pendulum	−130 ± 16	−581 ± 178	−564 ± 103	−142 ± 5	−153 ± 24	−937 ± 156

rewards, disregarding the reward at each step and thus avoiding the influence of the deceptive single reward in the environment. Due to the strength of EA on Swimmer, hybrid algorithms have the best performance in this environment. Although PDERL is also GA-based, its crossover and mutation operators are based on propagation and the features of its EA component may be relatively closer to gradient-based methods. In addition, as shown in Fig. 2, AERL obviously learns faster than ERL, which demonstrates that the proposed adaptive mutation and early termination strategies are more efficient than the corresponding approaches employed in ERL.

The Hopper and Walker2d environments are characterized by instability, leading to significant variations in an agent's performance across different evaluation episodes. TD3 shows a clear decline in its learning curve, making it important to protect excellent individuals from damage in these two environments. Population-based methods offer inherent advantages in this regard, as they typically store good individuals as elitists and directly include them in the next generation of individuals. As demonstrated in Fig. 2, hybrid algorithms exhibit greater stability in performance on these benchmarks. While TD3 shows faster learning in the early stages, our method catches up and ultimately surpasses its performance in later stages of learning.

The learning curves of various algorithms compared on four classic continuous control problems are depicted in Fig. 2 . The shaded areas indicate the standard variation in performance. Table 1 presents the mean and standard deviation of final performances across different test problems. Overall, our method achieves superior results on MountainCarContinuous, Pendulum and LunarLanderContinuous. On BipedalWalker, our results are slightly inferior to TD3. On MountainCarContinuous, the results of ERL and TD3 are zero, which is due to the sparse reward in this environment. If the agent cannot explore the top of hill to obtain the final positive reward, the agent will only receive a negative reward as the cost of action in every step. Therefore, the agent tends to remain stationary to avoid receiving a negative reward, which results in a cumulative reward of zero.

4.5 Ablation Experiments

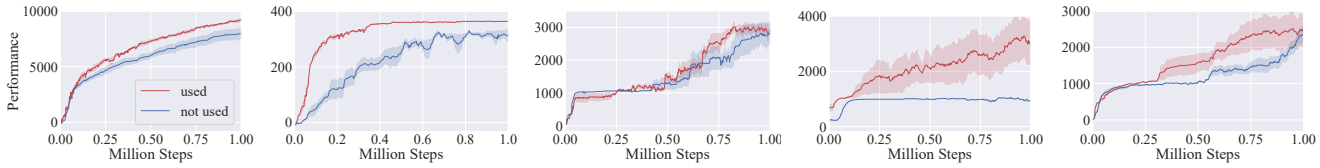
In this section, we conduct ablation experiments to validate the effectiveness of the proposed adaptive mutation operator and evaluation strategy.

First, we compare our method with fixed standard variation variants and we select the relatively challenging Walker2d environment to demonstrate the differences between the complete algorithm and its variants. The fixed σ values are set from 0.01 to 0.5. As shown in Fig. 3(b), our method outperforms all its variants with fixed σ. Besides, the algorithm achieves the second performance when $\sigma = 0.2$, not the largest or smallest value in all trials. This phenomenon illustrates that σ values that are too large or too small are not beneficial to the optimization of the population. Then, we record the variation of σ value caused by our adaptive mutation operator in multiple environments and plot it in Fig. 3(c). In the early period of learning, the values of σ change greatly. As the learning goes on, the values of σ show a dropping trend in principle. Besides, the variation curves of σ are different in various environments. This indicates that our method can adaptively control the value of σ in different periods and different environments.

Furthermore, we replace our early termination strategy with a standard evaluation function to enable a comparison with the original AERL. In this variant algorithm, each individual in the population undergoes a complete evaluation, and all of their experiences are stored in the replay buffer, which is identical to the evaluation function in ERL. We record the performance of both algorithms in five independent runs with different random seeds and plot the results in Figure 4. As shown in Figure 4, the original AERL consistently achieves notably better final performance than the variant that does not use the early termination strategy across all tested environments. In addition, by discarding irrelevant evaluations and experiences, our method exhibits greater efficiency and faster learning than its variant.

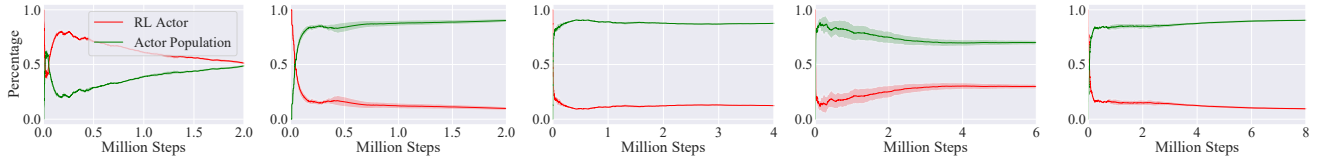
4.6 The Alternation between Learning and Evolution

Fig. 5 illustrates the percentage of timestep numbers of both the RL actor and actor population during the learning process of our method. In different environments or periods, the interaction step



(a) HalfCheetah (b) Swimmer (c) Hopper (d) Ant (e) Walker2d

Figure 4: The performance comparison between AERL and its variants without early termination strategy



(a) HalfCheetah (b) Swimmer (c) Hopper (d) Ant (e) Walker2d

Figure 5: The timestep number percentage of RL actor and actor population

numbers of the RL actor are varied. This is because the evaluation strategy in our method provides better-performing actors with greater opportunities to interact with the environment. In comparison, for ERL and PDERL, the ratio is almost $1 : N$ in all circumstances, where N represents the size of the population.

In the HalfCheetah environment, the percentage of the RL actor's values are the highest among all environments. Even in the final period, the proportion of the RL actor's timesteps remains at approximately 50%, indicating that the progress in performance primarily comes from the RL agent's learning. Conversely, in the Swimmer environment, the proportion of the RL actor is approximately 10% for most of the periods. As discussed in the preceding section, EAs consistently perform better than RL methods in this environment; hence, it is reasonable for the percentage of the RL actor to remain low. In unstable environments such as Hopper and Walker2d, the RL actor's performance can vary dramatically, leading to a relatively lower proportion of the RL actor. In the Ant environment, even though the percentage value of the RL actor is lower than that of the actor population, it is still approximately 30%. This observation demonstrates that the RL actor has more influence in Ant than in Walker2d or Hopper, which is more similar to the HalfCheetah environment.

5 CONCLUSIONS

In this paper, we have proposed an adaptive mutation operator and an early termination strategy for ERLs. The adaptive mutation operator maintains the strength of mutation within a reasonable range and does not rely on gradient computation or any computationally complex operations. Instead, it uses the performance of newly produced individuals, which can be conveniently calculated during the evaluation process. In addition, the early termination strategy avoids useless evaluation for poor-performing individuals and discards their experiences to enhance the learning efficiency of RL agents. Then, these two improvements are integrated with RL algorithms and form a novel algorithm called AERL. Moreover, several experiments are conducted to compare the practical performance of our method with other state-of-the-art algorithms in various continuous control tasks. The experimental results demonstrate the effectiveness of our algorithm.

Regarding future work, there is scope for improvement in the rules of our adaptive mutation operator, as the current rules are relatively simple. Developing a more fine-grained range of rules could enhance the performance of our method. Furthermore, our approach does not currently employ a crossover operator. Thus, designing a powerful crossover operator within our method is a possible avenue for future research. Additionally, beyond fitness-based selection, it may be worthwhile to explore selection methods that focus more attention on the diversity [5, 20].

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Funds for Distinguished Young Scholar under Grant 62325307, in part by the National Natural Science Foundation of China under Grants 62203308 and 62173236, in part by the Guangdong Regional Joint Foundation Key Project under Grant 2022B1515120076, and in part by the Shenzhen Science and Technology Program under Grants JCYJ20220531101411027 and 20220809141216003, and in part by the Scientific Instrument Developing Project of Shenzhen University under Grant 2023YQ019.

REFERENCES

[1] Chang Wook Ahn and Rudrapatna S Ramakrishna. 2003. Elitism-based compact genetic algorithms. *IEEE Transactions on Evolutionary Computation* 7, 4 (2003), 367–385.

[2] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. 2019. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680* (2019).

[3] Cristian Bodnar, Ben Day, and Pietro Lió. 2020. Proximal distilled evolutionary reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 3283–3290.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[5] Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth Stanley, and Jeff Clune. 2018. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. *Advances in neural information processing systems* 31 (2018).

[6] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation* 6, 2 (2002), 182–197.

[7] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. 2016. Benchmarking deep reinforcement learning for continuous control. In *International conference on machine learning*. PMLR, 1329–1338.

[8] Agoston E Eiben, James E Smith, et al. 2003. *Introduction to evolutionary computing*. Vol. 53. Springer.

[9] Scott Fujimoto, Herke Hoof, and David Meger. 2018. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*. PMLR, 1587–1596.

[10] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*. PMLR, 1861–1870.

[11] Nikolaus Hansen. 2016. The CMA evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772* (2016).

[12] Hado Hasselt. 2010. Double Q-learning. *Advances in neural information processing systems* 23 (2010).

[13] Shauharda Khadka and Kagan Tumer. 2018. Evolution-guided policy gradient in reinforcement learning. *Advances in Neural Information Processing Systems* 31 (2018).

[14] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[15] Brad L Miller, David E Goldberg, et al. 1995. Genetic algorithms, tournament selection, and the effects of noise. *Complex systems* 9, 3 (1995), 193–212.

[16] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.

[17] Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisỳ, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. 2017. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science* 356, 6337 (2017), 508–513.

[18] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J Andrew Bagnell, Pieter Abbeel, Jan Peters, et al. 2018. An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics* 7, 1-2 (2018), 1–179.

[19] Aloïs Pourchot and Olivier Sigaud. 2018. CEM-RL: Combining evolutionary and gradient-based methods for policy search. *arXiv preprint arXiv:1810.01222* (2018).

[20] Justin K Pugh, Lisa B Soros, and Kenneth O Stanley. 2016. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI* (2016), 40.

[21] Reuven Y Rubinstein and Dirk P Kroese. 2004. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation, and machine learning*. Vol. 133. Springer.

[22] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. 2017. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864* (2017).

[23] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[24] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. Deterministic policy gradient algorithms. In *International conference on machine learning*. Pmlr, 387–395.

[25] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O Stanley, and Jeff Clune. 2017. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567* (2017).

[26] Karush Suri. 2022. Off-Policy Evolutionary Reinforcement Learning with Maximum Mutations. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*. 1237–1245.

[27] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[28] Emanuel Todorov, Tom Erez, and Yuval Tassa. 2012. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 5026–5033.

[29] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 575, 7782 (2019), 350–354.

[30] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. 2016. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224* (2016).

[31] Darrell Whitley. 1994. A genetic algorithm tutorial. *Statistics and computing* 4, 2 (1994), 65–85.

[32] Lingxi Xie and Alan Yuille. 2017. Genetic cnn. In *Proceedings of the IEEE international conference on computer vision*. 1379–1388.

[33] Han Zheng, Jing Jiang, Pengfei Wei, Guodong Long, and Chengqi Zhang. 2020. Competitive and cooperative heterogeneous deep reinforcement learning. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*.

[34] Qingling Zhu, Xiaoqiang Wu, Qiuzhen Lin, Lijia Ma, Jianqiang Li, Zhong Ming, and Jianyong Chen. 2023. A survey on evolutionary reinforcement learning algorithms. *Neurocomputing* 556 (2023), 126628.