Evolutionary Reinforcement Learning with Action Sequence Search for Imperfect Information Games

Xiaoqiang Wu[a], Qingling Zhu[a], Wei-Neng Chen[b], Qiuzhen Lin[a,*], Jianqiang Li[a] and Carlos A. Coello Coello[c]

[a]College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China
[b]School of Computer Science and Engineering, South China University of Technology, Guangzhou, China
[c]Computer Science Department, CINVESTAV-IPN, Mexico City, Mexico

ARTICLE INFO

Keywords:
Evolutionary algorithm
Reinforcement learning
Evolutionary reinforcement learning
Imperfect information game

ABSTRACT

Deep Reinforcement Learning (DRL) has achieved remarkable success in perfect information games. However, when applied to imperfect information games like Contract Bridge, DRL faces challenges not only from unobservable partial information but also from the lack of efficient exploration. Although several Evolutionary Reinforcement Learning algorithms (ERLs) have harnessed Evolutionary Algorithms (EAs) to enhance exploration capabilities, the practical performance of EAs is limited either by the high-dimensional parameter space or possibly inaccurate guidance from value networks. In this paper, we introduce a novel ERL algorithm that employs the Particle Swarm Optimization (PSO) algorithm to search for superior action sequences, thereby aiding agents in exploring uncharted territory. We conduct the search in action space and evaluate action sequences through interactions with the environment to avoid the limitations of parameter space and value networks, respectively. The diverse experiences collected in the search and evaluation can boost the learning of the DRL agent. In addition, the action sequence search is executed only when the agent converges in local optima, which can reduce the overall cost of action evaluation and avoid influencing the optimization process of DRL. Through experimental comparisons conducted on Contract Bridge, our method demonstrates superior performance when compared with several state-of-the-art DRL and ERL algorithms. Furthermore, we utilize our method in the Bridge Competition of AAMAS 2023 Imperfect Information Card Games Competition and rank the first.

1. Introduction

Deep Reinforcement Learning (DRL) has made great achievement in the game of Go, chess and shogi [41, 42], which are perfect information games. For two-player imperfect information game like heads-up no-limit Texas hold'em, DRL also demonstrates super-human performance [30]. Nevertheless, in the case of Contract Bridge, a multi-player imperfect information game, DRL has not reached the level of top human professionals [35]. When addressing challenges within this domain, DRL encounters obstacles stemming from incomplete observable information, enormous state space, the lack of diverse exploration, etc [21, 35]. In addition, when training agents with self-play, it is easy for agents to become trapped in local optima because of the homogeneous partners and opponents [2]. Simultaneously, the lack of efficient exploration makes escaping local optima a formidable challenge.

Over the past few years, several methods have introduced Evolutionary Algorithms (EAs) to improve the exploration capabilities of DRL, called Evolutionary Reinforcement Learning algorithms (ERLs) [3, 19, 21, 34]. EAs are a type of population-based heuristic algorithm inspired by natural evolution and have demonstrated powerful search ability in a wide spectrum of optimization problems [5, 27, 47]. Furthermore, an increasing body of research has applied EAs to the field of Deep Learning and produced more promising results than traditional methods [25, 50]. As a branch of this emergent domain, ERLs typically employ policy networks or actions to form the population and optimize them iteratively by EAs. Throughout this process, the superior individuals generated by EA can boost the learning of the RL agent. In turn, the information learned by RL can contribute to the update of the population. Through the mutual promotion between EA and RL, ERLs generally can achieve superior performance compared to the original EA or RL.

However, the majority of ERLs primarily utilize EAs to directly optimize the weights of policy networks. The high-dimensional parameter space makes the search of EAs inefficient. To alleviate this problem, Proximal Distilled ERL (PDERL) [3] proposes more efficient crossover and mutation operators. In addition, ERL-Re^2 [19] proposes the two-scale representation and employs EAs to optimize the output layer only, reducing the search space of EAs. Nevertheless, the output layer of a neural network often has at least thousands of parameters, which still leaves a too large solution space for EAs. To avoid the problem of vast search space, some algorithms utilize EAs to optimize the actions of agents [29, 38]. These algorithms use value networks to evaluate the actions generated by EAs. This evaluation approach is intuitive, but the guidance provided by value networks may be inaccurate because of inadequate training during the optimization process or inherent errors from value estimation [16].

*Corresponding author
ORCID(s): 0000-0002-0228-8226 (Q. Zhu); 0000-0003-2415-0401 (Q. Lin)

In this paper, we propose an innovative method named Evolutionary Reinforcement Learning with Action Sequence Search (ERL-A2S), which avoids the issues of high-dimensional solution space and possibly incorrect guidance from value networks. ERL-A2S employs the Particle Swarm Optimization (PSO) algorithm to search for superior action sequences and provide diverse experiences for the RL agent. The dimension of action space is significantly smaller than that of parameter space, making it more suitable for EAs. Furthermore, diverging from prior methods, the generated action sequences are evaluated by the practical rewards in the interaction process rather than the estimation of value networks. This approach allows for a more accurate evaluation of action values. The search for a superior action sequence may require multiple episodes of interaction. It is inefficient and unnecessary to execute the action sequence search and the evaluation of the RL agent with the same frequency. Thus, to reduce the overall cost of action evaluation, the action sequence search is executed only when the agent converges in local optima. In addition, it can also avoid influencing the original optimization process of RL. Throughout the search process, any discovered superior action sequence is stored in the experience replay buffer. These diverse experiences are not from the homogeneous polices trained by RL and are expected to help the RL agent escape the local optima. To validate the efficiency of our method, we implement it with TD3 and compare it with state-of-the-art RL and ERL methods on Contract Bridge. The experimental results demonstrate that our method can train more competitive agents, while other compared methods tend to converge prematurely. Moreover, we utilize our method in the Bridge Competition of AAMAS 2023 Imperfect Information Card Games Competition and rank the first (http://www.jidiai.cn/compete_detail?compete=31).

The organization of this paper is as follows. Section 2 provides a brief introduction of some background including base information about EA and RL, RL in imperfect information games and ERLs. Following that, we elaborate on our method in Section 3. Section 4 presents an empirical analysis of the performance of our method on the Contract Bridge benchmark. Finally, in Section 5, we summarize our findings and some future directions.

2. Background

2.1. Reinforcement learning

Reinforcement learning can be modeled as a Markov Decision Process (MDP), defined by a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$. At each timestep t, the agent receives a state $s_t \in \mathcal{S}$ from the environment, and selects an action $a_t \in \mathcal{A}$ to perform. Subsequently, the environment provides the agent with the next state s_{t+1} and a reward r_t according to the transition function $\mathcal{P}(s_t, a_t)$ and reward function $\mathcal{R}(s_t, a_t)$, respectively. This interactive process repeats until the agent receives a termination state. The primary objective of RL algorithms is to discover the optimal policy that maximizes the cumulative discounted reward R_t. This reward is mathematically defined as $R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r_t$, where γ represents the discount factor, and T is the length of a single episode.

Deep Deterministic Policy Gradient (DDPG) [26] and its variant Twin Delayed DDPG (TD3) [16] are the most widely used RL algorithms in ERLs. To extend the concept of Deep Q-Network (DQN) to the continuous action space, DDPG employs a policy network μ to output continuous actions and a value network Q to approximate the action-value function. The value network is updated by minimizing the loss function as described below.

$$L = \frac{1}{N} \sum_{i} (r_i + \gamma Q^{'}(s_{i+1}, \mu^{'}(s_{i+1})) - Q(s_i, a_i))^2 \quad (1)$$

where N represents the batch size, $Q^{'}$ and $\mu^{'}$ are target networks. The policy network is updated by maximizing the state-action value estimated by Q:

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

where θ means the parameters of the network. Building upon DDPG, TD3 incorporates two distinct value function networks, utilizing the minimum between the two estimates to mitigate the issue of value overestimation. To further diminish cumulative estimation errors, TD3 adopts the delaying policy update strategy, reducing the frequency of updates for both the policy and target networks.

2.2. Evolutionary Algorithm

EAs are a type of population-based heuristic algorithm that mimics the biological evolution. EA typically maintains a population $P = \{p_1, p_2, ..., p_n\}$ consisting of n individuals. By iteratively evaluating and updating the individuals within the population, EAs search for optimal solutions to optimization problems. In most ERLs, the type of individuals is policy π or action a. For policy individual π, the fitness is generally calculated by $f(\pi) = \sum_{t=0}^{T} r_t|\pi$ [3, 19, 21, 34]. For action individual a in [29, 38], the fitness is calculated by $f(s, a) = Q(s, a)$.

PSO [33] is a swarm intelligence optimization algorithm inspired by biological behavior. In PSO, a population of N particles is updated iteratively with the guidance from both the historically best positions and the globally best position. The position of particle i is represented as vector X_i. The historically best position of particle i is denoted as $pBest_i$ and the globally best position $gbest$ is the best among all $pBest_i$. The update of particle velocity V_i is as follows:

$$V_i = wV_i + c_1 r_1 (pBest_i - X_i) + c_2 r_2 (gBest - X_i) \quad (3)$$

The particle position is updated by V_i:

$$X_i = X_i + V_i \quad (4)$$

where w, c_1 and c_2 are hyperparameters. r_1 and r_2 are random numbers. After each iteration of the population, $pBest_i$ and $gbest$ are updated through the comparison with new individuals.

2.3. Reinforcement learning in imperfect information games

DRL has been successfully applied to numerous imperfect information games. DeepStack [30] beats the professional poker players in two-player no-limit Texas hold'em. ReBeL [4] proposes a general RL framework for two-player zero-sum games and exhibits superhuman performance in heads-up no-limit Texas hold'em. SPARTA [23] achieves new state-of-the-art scores on the cooperative Game Hanabi using innovative search techniques. Suphx [24], integrating DRL with techniques such as global reward prediction, surpasses most top human players in Mahjong. Douzero [49] utilize the Monte-Carlo methods to enhance DRL and demonstrates superior performance than previous AI program in DouDizhu, which is a three-player imperfect information game with a large action space.

Nevertheless, DRL has not yet attained the level of top human professionals on Contract Bridge, which is a multi-player imperfect information game that involves both competition and cooperation. Over the past few years, several methods have applied DRL to the bidding of Contract Bridge. In [48], an agent is trained using DQN on a simplified bidding problem without competition. In [35], a hands estimation network is used to infer the hand of partner. Additionally, it combines supervised learning and RL to train bidding policies. Joint Policy Search (JPS) [45] improves the policies of the team jointly to achieve better practical performance. All these three algorithms are tested in a team competition setting and outperform a top computer bridge software in the theoretical results calculated by Double Dummy Analysis [17]. However, in the real world, there are not only team competitions, but also individual competitions. In this paper, we focus more on the evaluation by individual competitions, that is, each algorithm controls one agent and plays with the other three agents manipulated by different algorithms. In this way, each agent must learn to cooperate with unknown agents, which brings more difficulties to the collaboration between agents.

2.4. Evolutionary reinforcement learning

In recent years, many ERLs have been proposed to harness the strengths of methods from both domains. Among all these methods, the majority of ERLs prefer to leverage EAs to optimize policy networks. ERL [21] optimizes a population of policies by Genetic Algorithm (GA) and a separate policy by DDPG. The policy population generates diverse experiences for the training of RL, while the learned information of RL is shared with the population by inserting the RL policy into the population. Through the interaction between EA and RL, both methods mutually reinforce each other and achieve a better final performance. Despite demonstrating the superiority of hybrid methods, ERL's genetic operators are deemed destructive, limiting its performance. To address this issue, PDERL [3] introduces the proximal mutation and distillation crossover operators. Proximal mutation utilizes the sum of gradients to reduce the mutation strength of each weight, enhancing the stability of the mutation process. Distillation crossover exploits parents' experiences to train their child using a form of Behaviour Cloning [32]. These two operators exhibit increased stability compared to the original operators and improve the performance. To improve the sample efficiency and reduce the computational burden, a Surrogate-assisted Controller (SC) [46] is proposed to estimate the individual fitness. Subsequently, ERL-Re^2 [19] introduces a two-scale representation and utilizes GA to optimize the weights of the output layer, reducing the search space for GA. Building on the proposed two-scale representation, it also employs a value function approximator for fitness estimation. Furthermore, Collaborative ERL (CERL) [20] trains a population that consists of actors with different discount factors to alleviate the sensitivity of the method to hyperparameters.

The aforementioned methods typically utilize GA to evolve the policy population, and there are also several works that leverage the strengths of other methods. CEM-RL [34] introduces a distinct framework that integrates RL algorithms with the cross-entropy method [18, 36]. CEM-RL optimizes half of the population by RL and produces the next generation using the better half of the population. In comparison to the combination mechanism in ERL, this approach is better suited for the cross-entropy method, achieving improved performance over ERL. Evolution-based Soft Actor-Critic (ESAC) [44] employs a similar framework to ERL, but it uses Evolution Strategies (ES) and SAC as its fundamental components. Additionally, ESAC introduces Automatic Mutation Tuning to enhance the algorithm's robustness to hyperparameters. Moreover, Recruitment-imitation Mechanism (RIM) [28] introduces the imitation learning into the ERL framework and combines the strengths of three distinct methods.

In contrast to methods that evolve policies, several approaches employ EA to search for superior actions to promote the learning of RL algorithms. Cross-Entropy Guided Policies (CGP) [43] uses the CEM policy to replace the RL policy for action selection, which utilizes the CEM to search for the action with the maximum value. Additionally, the RL policy is trained by imitating the behavior of the CEM policy. Evolutionary Action Selection-TD3 (EAS-TD3) [29] utilizes PSO to evolve the action chosen by the policy network. Subsequently, the policy network learns the evolved action through a process similar to behaviour cloning. Self-guided and self-regularized actor-critic (GRAC) [38] leverages the actor network to output a Gaussian distribution of action first. Then, CEM is executed to search for an action with a higher Q value, which can be applied to expedite the learning of the RL agent.

Moreover, there are several distinct methods that evolve interpretable decision trees [7, 13, 14, 40]. In [11] and [12], Grammatical Evolution (GE) and Genetic Programming (GP) are employed to optimize the decision trees for solving control tasks and image-based Atrai benchmarks, respectively. In [10], the optimization problem is decomposed into two sub-problems to combine the advantages of GE and Q-learning. In this scheme, GE is responsible for

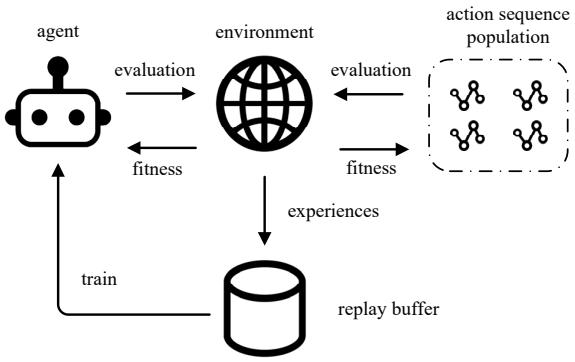


Fig. 1: The general framework of ERL-A2S.

decomposing the state space while the RL method is responsible for finding an action for the corresponding subspace. Subsequently, in [8], the scheme of two-level optimization introduced in [10] is extended into the multi-agent domain. In addition, different coevolutionary schemes for reducing the number of parallel processes are studied and discussed. Apart from goal-directed methods, Quality Diversity methods are utilized in [15] to search diverse solutions and improve the exploration capabilities.

Algorithm 1: Evolutionary Reinforcement Learning with Action Sequence Search

Input: population size n;
maximum timesteps m;
sample size L;
Initialize policy network π and other networks with random values;
Initialize an empty replay buffer B and timestep t;
while $t < m$ **do**
  $fitness, step =$ Evaluation(π, B) (Algorithm 3);
  $t = t + step$;
  Calculate average return in recent L and $2L$ episodes R_L, R_{2L};
  if $R_L < R_{2L}$ **then**
    $step =$ Action_Sequence_Search($fitness$, B, n) (Algorithm 2);
    $t = t + step$;
  Execute gradient update in RL method;

3. The proposed algorithm

This section introduces the details of our proposed ERL-A2S algorithm. The general framework of ERL-A2S is illustrated in Fig. 1. In each iteration of the algorithm, the RL agent interacts with the environment and learns from its experiences. Once the agent converges to local optima, the population is initialized by the action sequence of the RL agent in this iteration. Then, the evolutionary search is conducted until a superior action sequence is found or the maximum number of generations is reached. The diverse experiences from superior action sequences are stored in the replay buffer and used in the subsequent training. After a number of such iterations with evolutionary search, the RL agent can learn novel skills from diverse experiences and continue its own optimization until it becomes stuck in local optima again.

Algorithm 2: Action Sequence Search

Input: fitness of RL agent f_{RL};
replay buffer B;
population size n;
Output: elapsed timesteps t
Obtain last action sequence of RL agent S_{RL};
Initialize population $P = \{S_1, S_2, ..., S_n\}$, where $S_i = S_{RL} + N(0, \sigma)$;
Initialize a temporary buffer T, $t = 0$;
Initialize the personal best $pBest$, global best $gbest$, velocity V;
while $generation < max\_generation$ **do**
  for $i = 1$ *to* n **do**
    $V_i = wV_i + c_1 r_1 (pBest_i - S_i) + c_2 r_2 (gBest - S_i)$;
    $S_i = S_i + V_i$;
    $fitness, step =$ Evaluation(S_i, T);
    Update $pBest_i$ and $gBest$ by $fitness$;
    $t = t + step$;
    if $fitness > f_{RL}$ **then**
      Append T to B;
      break;
    Empty the buffer T;
return t;

Algorithm 3: Evaluation

Input: actor π or action sequence S; replay buffer B;
Output: $fitness$, elapsed timesteps t
Initialize $fitness = 0$, $t = 0$;
Reset environment and get initial state s_0;
while *env is not done* **do**
  Select action $a_t = \pi(s_t|\theta^\pi) + noise$ or $a_t = S_t$;
  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 B;
  $t = t + 1$
return $fitness$, t;

3.1. The complete algorithm of ERL-A2S

We combine our action sequence search with RL algorithms to compose a novel algorithm called ERL-A2S. At the beginning of the algorithm, the agent is optimized by RL

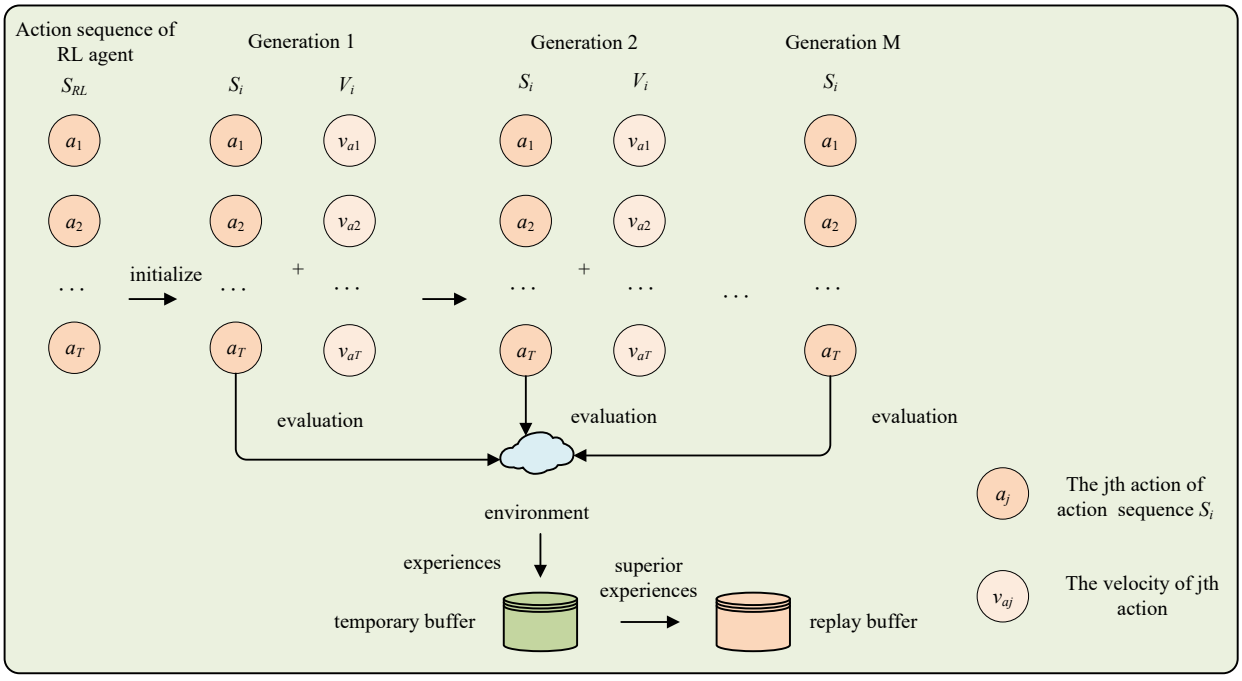


Fig. 2: The general process of action sequence search.

methods only. Once the algorithm detects that the agent possibly converges to local optima, the action sequence search is executed. When the agent reaches convergence, it tends to act in a fixed manner and only acquires homogeneous experiences from the interaction with the environment. In this scenario, it is difficult for the RL agent to learn new skills on its own. Therefore, we employ the action sequence search to obtain superior action sequences and provide diverse experiences for the learning of the RL agent. Once the RL agent gets out of the local optima, the part of evolutionary search will be skipped, which can reduce the unnecessary cost of interaction.

As illustrated in Algorithm 1, the major part of the method is an iterative process. First, the RL agent interacts with the environment and acquires experiences from the interaction (line 4). At the same time, the sum of rewards during the interaction is regarded as the fitness of the agent. Then, we assess whether the agent has converged by calculating and comparing its average return in recent L and $2L$ episodes (lines 6-7). In practical experiments, we set the hyperparameter L to 10^5, which is enough to determine the performance of the agent. If the average return in recent L episodes is less than the average value in recent $2L$ episodes, the evolutionary search is executed to obtain diverse experiences for the optimization of the RL agent (line 8). Finally, the neural network models are updated by the RL method (line 10). It is noteworthy that, although the timesteps used by A2S are accumulated in the total timesteps, the number of gradient updates is equal to the number of timesteps used by the RL agent, excluding the timesteps used by A2S. This can avoid excessive updates with the old data. The entire iterative process will be terminated when the number of elapsed timesteps is not less than the maximum timesteps m, which is a predefined variable that controls the maximum number of interactions for the training.

3.2. Action sequence search

The aim of the search is to discover superior action sequences and provide diverse experiences to promote the further learning of the RL agent. In contrast to previous methods, our method eliminates the need to cope with the high-dimensional space of network parameters and avoids incorrect guidance from the estimation errors of value networks. In addition, the search is conducted only when the RL agent converges to local optima, which can reduce the interaction costs of searching action sequences. The general process of the action sequence search is depicted in Fig. 2. As this chart shows, the population consists of action sequences and is initialized based on the action sequence from the RL agent. Then, each individual is updated by PSO and evaluated by interacting with the environment. We select PSO for two main reasons. Firstly, PSO has been utilized for searching superior actions and has demonstrated promising results in prior studies [29]. Secondly, PSO has been widely applied across various fields and it does not impose a significant computational burden. The experiences of each individual are stored in a temporary buffer and only the experiences of superior action sequences are stored in the formal replay buffer. The search is terminated when the number of generations reaches the maximum value.

The pseudocode of our method is provided in Algorithm 2. First, the population is initialized by adding Gaussian noises to the action sequence of the RL agent (line 2).

Initializing the population by the RL action sequence can enhance the search efficiency while introducing random noises can generate different new individuals. To some extent, this initialization makes a trade-off between the search efficiency and the diversity. Then, other variables are initialized (lines 3-4). Following the initialization, the population is iterated until the termination criterion is satisfied. In each iteration, we first update the velocity and position of particles according to Eq. (3) and Eq. (4) (lines 7-8). In our algorithm, the positions of particles represent the action sequences. After that, the action sequences are evaluated and the fitness values are obtained by calculating the cumulative rewards during the interaction (line 9). To precisely evaluate the action sequences, the initial state s_0 is equal to the corresponding initial state of S_{RL} and remains unchanged during the search. The experiences of individuals are stored in the temporary buffer T. Finally, if the fitness of the individual is larger than that of the RL agent, the experiences in temporary buffer T will be stored in the formal replay buffer B and the iteration will be terminated (lines 12-14). Otherwise, the buffer T will be emptied (line 15) and the iteration will be continued.

4. Experimental studies

4.1. Benchmark problems

In this study, we utilize the environment in the Bridge Competition of AAMAS 2023 Imperfect Information Card Games Competition as the benchmark problem to assess the performance of compared algorithms. The source code of the benchmark problem can be found at https://github.com/jidiai/Competition_AAMAS2023. In this benchmark, each algorithm controls one agent and plays with the other three players, which includes a partner and two opponents. Given that the control methods for the other players are unknown, the agent must learn to cooperate and compete with different players. Same as prior studies [35, 45, 48], we mainly focus on the bidding phase, which is recognized as the hardest part of Contract Bridge [35].

Contract Bridge is a four-player imperfect information game that involves cooperation and competition. The four players are divided into two teams: North and South, East and West. At the beginning of each Game, each player randomly obtains 13 cards from a standard deck of 52 cards, which has 4 suits (club ♣, diamond ♢, heart ♡ and spade ♠) and 13 ranks (from 2 to A). The subsequent process can be separated into two phases: the bidding phase and the playing phase. In the bidding phase, each player tries to communicate with its partner and bid the best contract for their hands. A valid contract comprises a level (ranging from 1 to 7) and a suit (one of the 4 suits or NoTrump). The player must bid a larger contract than the last one. Otherwise, the player can only pass (the double and redouble are not considered here, because they have no effect in the benchmark). The order of all contracts is {1♣, 1♢, 1♡, 1♠, 1NT, 2♣, ...,7NT}, where NT represents NoTrump. After three consecutive passes, the largest contract is selected as the contract of this game and the suit of this contract becomes the trump suit. The cards of the trump suit are trump cards, which are larger than the cards of other suits. NoTrump means that there are no trump cards in this game. In the team that bids the final contract, the player who first bids the trump suit is called the declarer.

After the bidding phase, the playing phase begins. This phase consists of 13 rounds. In each round, each player must play one card and the one who plays the largest card wins this trick. The player to the left of the declarer leads the first round, that is, playing first. Once the first card has played, the partner of the declarer, called the dummy, opens its hand to all players. The subsequent rounds are led by the player that wins the last trick. During the playing, each player must first play the card of the same suit as the leader's card. Only when there is no card of the same suit, the player can play cards of other suits. In this phase, the aim of all players is to take as many tricks as possible.

4.2. Performance Metric

To make the contract, the declarer side needs to take $X + 6$ tricks, where X is the level of contract. For instance, if the North bids 1♡, its team must take at least 7 tricks to complete the contract. In the benchmark, if the declarer side makes the contract, its reward will be $X + 8$. Otherwise, its reward will be $Y - X - 6$, where Y is the number of tricks that it takes. For the other team, called the defender side, its reward equals the number of tricks it takes. To ensure a fair comparison between RL algorithms and population-based algorithms, the timesteps used by each individual of the population are accumulated. In addition, we pick the best model during the entire training process to measure each algorithm's performance.

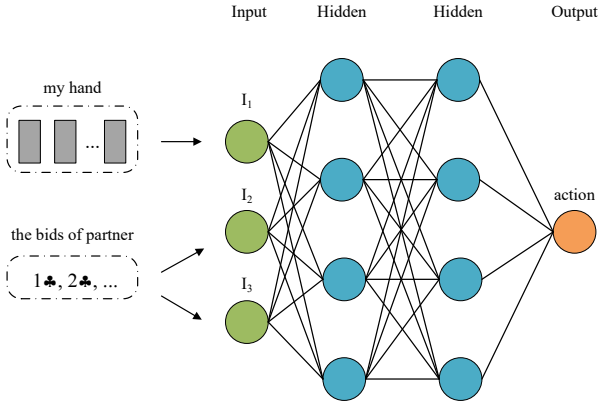


Fig. 3: The network architecture of the policy network.

4.3. Experimental settings

To assess the practical efficacy of the proposed algorithm, we conduct a comparative analysis against state-of-the-art ERLs (ERL [21], PDERL [3], ERL-Re^2 [19]), as well as RL algorithms (TD3 [16] and PPO [37]). We utilize the implementation published by the authors for all compared algorithms. The network architectures and parameter settings in each algorithm are set as suggested in the corresponding

Table 1
The hyperparameters for all compared algorithms. × represents that the parameters are not used in algorithms.

Hyperparameters	ERL-A2S	ERL	PDERL	ERL-Re^2	TD3	PPO
Discount factor	0.99	0.99	0.99	0.99	0.99	0.99
Actor learning rate	3e-4	5e-5	5e-5	1e-3	3e-4	3e-4
Critic learning rate	3e-4	5e-4	5e-4	1e-3	3e-4	3e-4
Target weight	5e-3	1e-3	1e-3	5e-3	5e-3	×
Batch size	256	128	128	128	256	32
Replay buffer size	1e6	1e6	1e6	1e6	1e6	×
Actor network architecture	256,256	128,128	128,128	400,300	256,256	64,64
Critic network architecture	256,256	400,300	400,300	400,300	256,256	64,64
Population size	5	10	10	5	×	×
Mutation probability	×	0.9	0.9	0.9	×	×

papers and some critical hyperparameters are listed in Table 1. The network architecture used in all compared algorithms is a multi-layer perceptron with 2 hidden layers and the numbers of neurons in hidden layers are listed in Table 1. All algorithms are trained for 6 million timesteps. The unique hyperparameters used in our method are detailed as follows. The inertia weight w is decreased linearly from 0.9 to 0.4 and both acceleration coefficients c_1 and c_2 are 2 in PSO, as the settings in [39]. The maximum number of generations in each search is 5. The sample size L is 10^5. The range of particle positions is set from 0 to 8 (exclusive) to align with the range of actions (the level of contract), while no restrictions are imposed on velocity range. The PSO used in our method is the classical version because it can meet the requirements of finding superior actions. A comparable analysis of the hyperpameters is provided in section 4.7.

In the bidding phase, the information observed by each agent includes its hand and the bidding of all players. If directly encoding the observation information by 0-1 vector, the dimension of the input representation is at least 192 (52 for hand and 4x35 for the bidding of all players). To learn from such an input representation, the policy network often needs to utilize 8 or more hidden layers [35, 45], which brings many difficulties for the training. Therefore, We simplify the input representation of neural networks to enable the MLP with 2 hidden layers to learn a superior policy, which is also the scheme that we used in the competition. The architecture of our policy network is illustrated in Fig. 3. Our input representation employs three scalar values to abstract the useful information. The first value I_1 is used to measure the strength of our hand, which can be calculated as follows:

$$I_1 = \sum_{i=1}^{13}(p_i - 10) + \sum_{j=1}^{4}(q_j - 5) + \sum_{j=1}^{4}(3 - min(q_j, 3)) \quad (5)$$

where p_i represents the point of each card in hand (the points of J-A are 11-14) and q_j is the number of cards in each suit. In this equation, the first term can measure the strength of card points while the latter two terms measure the strength of the card distribution. The second value I_2 represents the largest level of the partner's bids, which is used to assess the strength of the partner's hand and can be expressed as follows:

$$I_2 = max\{level_i \mid level_i \in S_{level}\} \quad (6)$$

where S_{level} is the level part of the partner's bidding sequences and $level_i$ is the level of the partner's i-th bid. The last value I_3 is the number of cards in the suit of the partner's last bid, serving as an assessment of the strength of the partner's bid for our team. This can be expressed as follows:

$$I_3 = \sum_{i=1}^{13} x_{ij} \quad (7)$$

where j is the suit of the partner's last bid and x_{ij} is a binary variable that equals 1 when the i-th card in hand belongs to suit j, and 0 otherwise. To ensure a fair comparison, the input representations used by all compared algorithms are the same.

An action consists of two parts: a suit and a level. The value of suit is determined by the strength of cards in each suit and only the level is searched by PSO. The level is an integer which ranges from 0 to 7. Therefore, the range of the particle position is from 0 to 8 (exclusive). And the position (a real number) will be floored to obtain the action (an integer). By discretizing the actions, our method can also be applied to the discrete action space.

In the test problems, the action sequence is variable-length, because the length of bidding sequence varies in different games. In the search of PSO, the action sequence is fixed-length and the length is equal to the length of RL policy's action sequence. When evaluating individuals(action sequences) in population, the step in the interaction process may exceed the maximum length of the sequence. In this situation, we use RL policy to assist the individual for selecting subsequent actions, and the results of evaluation can be seen as an estimation of real fitness.

All algorithms are trained by self-play and the opponents are the copies of the current agent. To accelerate the training and minimize the impact of playing on the results of bidding, we utilize the Double Dummy Solver (DDS) [17] to obtain the final result of the game during training. DDS calculates

Table 2
The average results of different team combinations against diverse opponents. Each value represents the average results of the team that includes two different agents against other team combinations.

	ERL-A2S	ERL	PDERL	ERL-Re^2	TD3	PPO
ERL-A2S	/	5.78	6.28	6.08	**6.41**	6.05
ERL	**5.78**	/	5.42	5.24	5.61	5.37
PDERL	6.28	5.42	/	5.88	**6.33**	5.87
ERL-Re^2	**6.08**	5.24	5.88	/	6.01	5.56
TD3	**6.41**	5.61	6.33	6.01	/	5.98
PPO	**6.05**	5.37	5.87	5.56	5.98	/
average	**6.12**	5.48	5.96	5.75	6.07	5.77

the maximum number of tricks that the declarer side can take based on the assumption that each player can see the hands of others. It has been demonstrated as a good approximation to the expert playing [35]. Consequently, there is no playing phase during training and the results obtained from DDS are used as the results of bidding.

The training for all algorithms is carried out on Ubuntu 18.04.6 LTS operation system, NVIDIA Tesla V100 GPU, Intel(R) Xeon(R) Gold 6126 CPU and 256G memory. The comparative experiments on the benchmark are conducted on the Windows 10 operation system, Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz CPU, and 16G memory.

4.4. Results for cooperating with unknown agents

We first conduct experiments to test the algorithms' performance for cooperating with unknown players. In these experiments, agents from six algorithms are combined to form fifteen different teams and all combinations confront each other. Each result of rivalries between two teams is the average value of 1000 games. Table 2 presents the performance of different team combinations against other combinations. For example, the value in the row 'ERL', column 'TD3' represents the average results of the team that consists of an ERL agent and a TD3 agent against the other 14 teams. The detailed results of each rivalry between teams are provided in Table 3.

As shown in Table 2, our method demonstrates superior performance even when teamed up with unknown partners. Specifically, our method surpasses other algorithms by a margin of at least 0.16 tricks per game, except for TD3. Our method only outperforms TD3 slightly because it can obtain many scores from weaker teams. If only the results of rivalries between our method and TD3 are considered, our method outperforms it by a margin of 0.32 tricks per game when the partner is the same (The average results of AB vs BE, AC vs CE, AD vs DE and AF vs EF in Table 3).

When cooperating with the agents of ERL, ERL-Re^2, TD3 and PPO, our method shows the best performance among all compared algorithms. For PDERL, TD3 shows better team performance than our method when competing with 15 different teams. Nevertheless, as shown in Table 3, the team 'AC' (including agents of ERL-A2S and PDERL) outperforms the team 'CE' (including agents of PDERL and TD3) by 0.61 tricks per game in the rivalries between both teams.

4.5. Results for cooperating with familiar agents

In this section, we first conduct experiments to test the algorithms' performance in a situation where the agents can cooperate with familiar partners. In the experiments, each algorithm controls one team against other teams and the average value of 10000 games is recorded as the reported result.

As shown in Table 4, our method exhibits a significant performance advantage over other compared algorithms when each algorithm controls a team. Although the difference of average results for competing with all methods between TD3 and our method is 0.06 tricks per game, our method surpasses TD3 by a margin of 0.22 tricks per game in 10000 games between both teams. In the games between the same algorithms, the result of our method is slightly below the results of TD3 and PDERL. However, in the rivalries between ERL-A2S and these two methods, our method outperforms them by a margin of 0.56 and 0.22 tricks per game, respectively. This phenomenon indicates that our agent is more aggressive than others and can restrict the opponent teams' scores. By the comparison of TD3, it can be easily inferred that the skills of more accurate and aggressive bidding are learned from experiences of the evolutionary search, which validates the effectiveness of our method.

Then, we conduct experiments to compare our models in multiple runs with the best model of TD3. To demonstrate the differences in performance among models, each model controls a team against the team controlled by the best model of TD3. In addition, the reported results are the average values of 1000 games.

As illustrated in Table 5, except for scores and relative scores, we also record our method's proportion as the declarer side and the success rate for making the contract. Overall, our method outperforms TD3 by a margin of 0.37 tricks per game. In addition, there are some differences among the models in different runs. Some models (1, 2, and 3) exhibit more aggressive behavior (with higher declarer rates but lower success rates) and significantly limit the opponent's scoring, while other models (0 and 4) are more conservative and precise (with higher success rates and scores but lower declarer rates).

Table 3
The detailed results of rivalries between different team combinations. A-F represents ERL-A2S, ERL, PDERL, ERL-Re^2, TD3 and PPO, respectively. Each value represents the average results of the team of the column against the team of the row.

	AB	AC	AD	AE	AF	BC	BD	BE	BF	CD	CE	CF	DE	DF	EF
AB	/	6.14	5.82	6.41	5.71	5.29	5.25	5.38	5.36	5.77	6.14	5.74	6.08	5.63	5.98
AC	5.72	/	6.01	6.01	5.74	5.26	5.01	5.39	5.35	5.58	5.82	5.59	5.72	5.26	5.84
AD	5.84	6.09	/	6.30	6.17	5.42	5.07	5.46	5.19	5.62	6.33	5.81	5.97	5.45	5.97
AE	5.22	6.17	5.67	/	5.67	5.16	4.92	5.51	5.05	5.48	6.00	5.49	5.65	5.16	5.68
AF	5.80	6.32	5.80	6.43	/	5.69	5.41	5.44	5.12	5.78	6.24	5.75	6.03	5.54	5.90
BC	5.84	6.04	6.20	6.48	5.88	/	5.57	5.78	5.56	6.15	6.44	5.91	6.16	5.67	6.16
BD	5.94	6.42	6.40	6.65	5.99	5.51	/	6.01	5.61	6.20	6.39	6.10	6.17	5.96	6.29
BE	5.80	5.94	6.09	6.09	6.07	5.31	5.06	/	5.30	5.96	6.41	5.73	5.97	5.46	5.78
BF	5.91	6.21	6.31	6.56	6.54	5.39	5.43	5.76	/	6.04	6.56	6.40	6.32	5.83	5.61
CD	5.85	6.30	6.16	6.55	6.13	5.55	5.37	5.57	5.57	/	6.49	5.87	6.10	5.78	6.05
CE	5.69	6.43	5.89	6.10	6.01	5.22	4.97	5.31	5.17	5.58	/	5.70	5.79	5.27	5 78
CF	6.05	6.43	6.19	6.69	6.32	5.55	5.49	5.70	5.37	6.21	6.49	/	6.18	5.62	6.35
DE	5.50	6.39	5.98	6.42	6.02	5.27	5.02	5.54	5.19	5.90	6.41	6.03	/	5.64	6.03
DF	5.89	6.56	6.43	6.69	6.39	5.78	5.51	6.03	5.51	6.17	6.61	6.32	6.14	/	6.34
EF	5.84	6.43	6.14	6.37	6.13	5.46	5.33	5.68	5.78	5.94	6.25	5.75	5.91	5.56	/
average	5.78	6.28	6.08	6.41	6.05	5.42	5.24	5.61	5.37	5.88	6.33	5.87	6.01	5.56	5.98

Table 4
The average results of direct rivalries between algorithms. Each value represents the average results of the team controlled by the algorithm of the column against the team controlled by the algorithm of the row.

	ERL-A2S	ERL	PDERL	ERL-Re^2	TD3	PPO
ERL-A2S	**6.06**	4.78	5.80	5.08	5.97	5.07
ERL	**6.63**	5.22	6.52	5.72	6.61	5.88
PDERL	6.36	5.06	6.14	5.44	**6.46**	5.49
ERL-Re^2	**6.93**	5.59	6.56	5.90	6.84	6.06
TD3	**6.19**	4.87	5.80	5.15	6.08	5.18
PPO	**6.87**	5.35	6.57	5.82	6.77	5.99
average	**6.51**	5.15	6.23	5.52	6.45	5.61

In addition to comparing our model with learning-based agents, we also conduct experiments to compare it with the rule-based agent. In the real world, bidding systems in Contract Bridge comprise two main branches: the natural system and the artificial system. The Standard American Yellow Card (SAYC) [1], created by the American Contract Bridge League (ACBL), is one of the most widespread natural systems. Based on the bidding rules in [1], we implement a rule-based agent and conduct experiments to compare our model with this agent. We first demonstrate the difference between the two agents by the opening bids (SAYC provides detailed regulations regarding the opening bids), as shown in Table 6. The opening bids data of our method is collected from practical games. As indicated in Table 6, our model is significantly more aggressive than the agent based on SAYC. For example, under SAYC, players generally need more than 12 HCP to bid a one-level contract, while our model only needs at least 3 HCP.

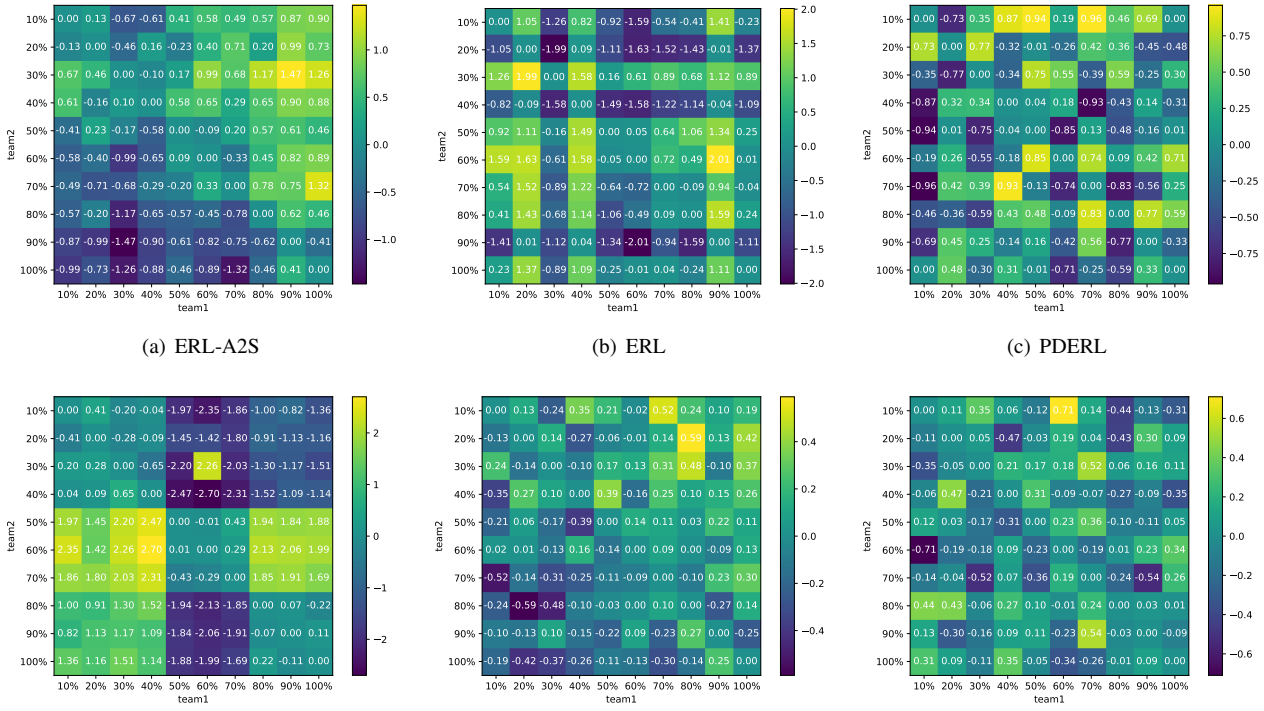
After that, we compare our model with this rule-based agent and provide the average results of 1000 games in Table 7. As shown in Table 7, our model significantly outperforms the rule-based agent. In addition, our model is more aggressive (higher proportion as the declarer side) and more precise (higher success rate with high declarer rate). One

Table 5
The average results of ERL-A2S (multiple runs with different random seeds) against TD3.

	seed0	seed1	seed2	seed3	seed4	mean	std.
Scores	6.20	5.98	6.03	5.94	6.27	6.08	0.13
Relative scores	0.23	0.44	0.47	0.46	0.24	0.37	0.11
Declarer rate	54.5%	69.6%	68.0%	65.9%	51.3%	61.9%	7.48%
Success rate	82.0%	71.1%	70.7%	69.0%	82.1%	75.0%	5.82%

Table 6
The opening bids of ERL-A2S's model and SAYC. HCP represents High Card Points (4 for A, 3 for K, 2 for Q and 1 for J.).

Opening bids	ERL-A2S	SAYC
1♣	3-15HCP, 4-6♣	12+HCP, 3+♣
1♢	3-15HCP, 4-6♢	12+HCP, 4+♢ or 4♠4♡3♢2♣
1♡	3-15HCP, 4-6♡	12+HCP, 5+♡
1♠	3-15HCP, 4-6♠	12+HCP, 5+♠
2♣	5-16HCP, 5-7♣	22+HCP
2♢	5-16HCP, 5-7♢	5-11HCP, 6+♢ / good 5♢ / poor 7♢
2♡	4-16HCP, 5-7♡	5-11HCP, 6+♡ / good 5♡ / ppor 7♡
2♠	4-16HCP, 5-7♠	5-11HCP, 6+♠ / good 5♠ / poor 7♠



(a) ERL-A2S (b) ERL (c) PDERL

(d) ERL-Re^2 (e) TD3 (f) PPO

Fig. 4: The relative results of rivalries between the policy models in different stages. For example, '10%' represents the model at 0.6M timesteps and '100%' represents the model at the 6M timesteps. Each value represents the relative results of the team controlled by the model of the column against the team controlled by the model of the row.

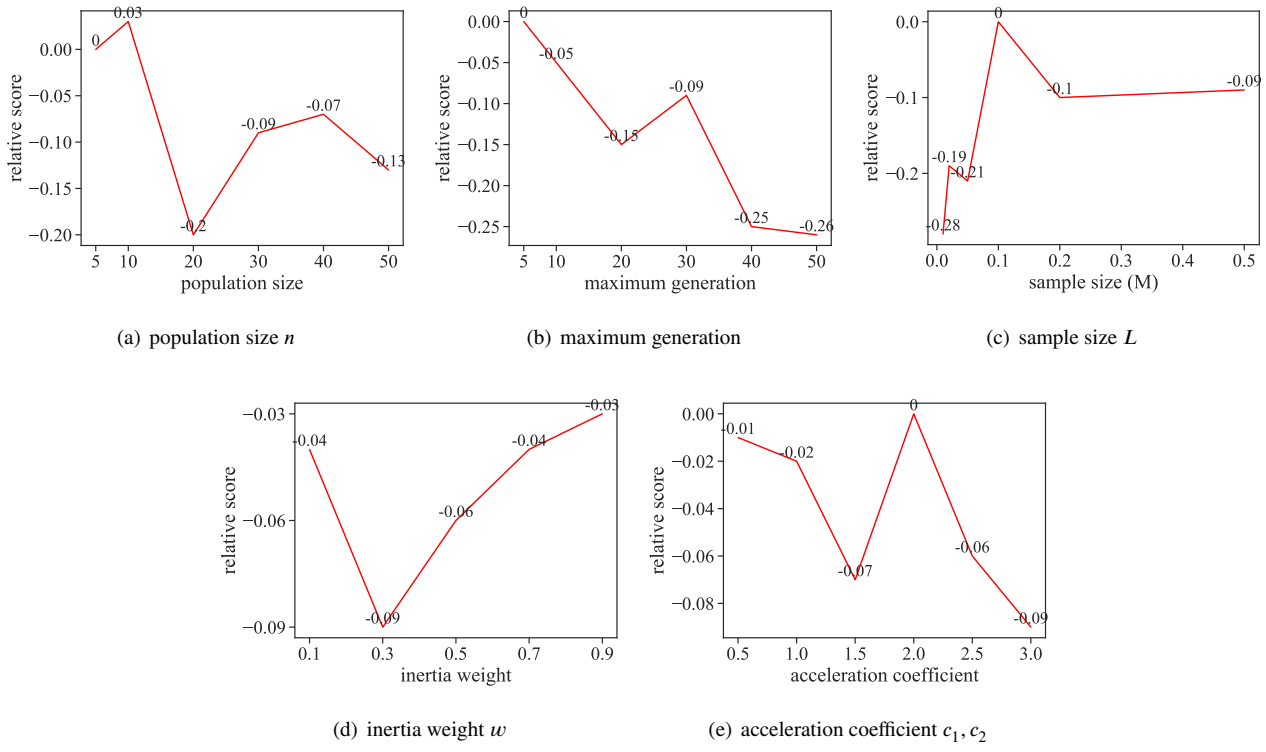
Table 7
The average results of ERL-A2S's model against the rule-based agent.

	ERL-A2S	rule-based agent
Scores	6.51	5.49
Relative scores	1.02	-1.02
Declarer rate	60.2%	39.2%
Success rate	82.2%	79.6%

important reason is that SAYC only provides general rules and human players generally need to adjust their strategies to adapt to different scenarios. However, it's impractical to design a rule for each possible situation. This is also one of the main advantages of the learning-based agent, which can learn a generic strategy to cope with all situations.

4.6. The performance in different training stages

When training with self-play, because the models of opponents in different training stages are different, it is difficult to observe the genuine performance changes from the training curves. Even if the same baseline is used to test the performance of models, it can only show the ability to play against a single model. To demonstrate the genuine performance change of policy models during training, we provide the relative results of rivalries between the policy models in different stages. Each result is the average value of 1000 games.



(a) population size n (b) maximum generation (c) sample size L

(d) inertia weight w (e) acceleration coefficient c_1, c_2

Fig. 5: The relative results of variants with different hyperparameters against the original ERL-A2S.

As shown in Fig. 4, the brighter the color of the column, the more superior the model is compared to others. In Fig. 4 (a), the last few columns are significantly brighter than the preceding columns, which means that the latter models outperform the previous models. While for other algorithms, there are several blue blocks at the top of the last column, which means that even the models at 90% or 100% timesteps show worse performance in the competition against models at 10% timesteps or 20% timesteps. This situation indicates that the models of these algorithms converge prematurely to the local optima and the subsequent training does not bring significant benefits. Compared to these methods, our model's performance shows significant improvement in the subsequent training, which can be attributed to the action sequence search. By introducing this evolutionary search, our agent can acquire diverse experiences and has a stronger ability to escape local optima.

4.7. Hyperparameters analysis

In this section, we conduct comparative experiments to demonstrate the impact of certain hyperparameters, which include the population size n, the maximum generation, the sample size L, inertia weight w and acceleration coefficient c_1, c_2. We utilize the relative scores of the variants with different hyperparameters against the original algorithm to measure their performance. All reported results are the average values of 1000 games.

First, we test the variants with population sizes of 10, 20, 30, 40 and 50. As shown in Fig. 5 (a), the performance changes relatively slightly when the population size increases, which indicates that the population size has a minor impact on the results. However, a larger population requires more interaction for evaluation. Therefore, the population size of 5 or 10 may be promising for our method. Then, we investigate the maximum number of generations of each search in experiments. The results show that increasing the maximum generation cannot bring benefit to the performance. Because the search is executed for many times during the training, excessive search may waste some resources. In addition, the choices of RL policy may be the best and there are no superior action sequences. In this situation, the additional search serves no purpose. For the sample size L, we test 0.01M, 0.02M, 0.05M, 0.1M, 0.2M and 0.5M. As shown in Fig. 5 (c), too small or large values of L are not the best choices for our method. The former may not be sufficient to determine the status of the agent, which makes the evolutionary search impede the learning of the agent. The latter slightly decreases the performance because it may limit the use of A2S. Finally, we investigate the impact of inertia weight and acceleration coefficient. As depicted in Fig. 5 (d) and (e), compared to other hyperparameters, our method is less sensitive to the inertia weight and acceleration coefficient. The performance of most variants is only slightly inferior to that of the original method.

4.8. Time consumption analysis

Except for the performance analysis, we also conduct experiments to compare the runtime of ERL-A2S and other

Table 8
The time consumption of algorithms for every 10000 timesteps.

Algorithms	ERL-A2S	ERL	PDERL	ERL-Re^2	TD3	PPO
Seconds	58.1	201.4	362.8	404.5	70.1	28.7

algorithms. The experiments are run on NVIDIA GeForce RTX 4090 and Intel(R) Xeon(R) Platinum 8336C CPU. The process of each algorithm is run alone on a single GPU and is allocated enough CPU resources during experiments.

As shown in Table 8, our method even runs faster than the vanilla TD3. This is because TD3 executes gradient updates at each timestep, but our method only executes it at the timesteps used by the RL agent. Therefore, our method needs fewer gradient updates than TD3. In addition, the evolutionary search is only conducted when the RL agent converges to the local optima, which only brings minimal computational burden compared to gradient descent. PPO is the fastest because it only executes limited gradient updates after collecting a batch of tuples. Other ERLs consume more time because of their evolutionary operations on neural networks. In comparison to ERL, the extra time of PDERL and ERL-Re^2 primarily stems from the gradient computation for the genetic operators and fitness estimation respectively.

5. Conclusions

In this paper, we propose ERL-A2S, a novel evolutionary reinforcement learning algorithm with action sequence search. This method utilizes PSO to search superior action sequences to provide diverse experiences for the learning of the agent. The introduction of this evolutionary search enhances the exploration capability of the RL method. Moreover, the search conducted in the action space and the evaluation by the interaction with the environment avoid the limitations of high-dimensional parameter space and value networks, respectively. In addition, we use it selectively, only when the RL agent converges prematurely, thereby reducing the overall computational cost of the search. We implement ERL-A2S with TD3 and conduct several experiments to compare its practical performance with other state-of-the-art ERLs on the Contract Bridge. The experimental results demonstrate the effectiveness of our algorithm. Furthermore, we investigate the performance of agents in different training stages and conduct hyperparameters analysis, which further validates the efficacy of our method.

Regarding the limitations and future work, similar to previous ERL methods, although we validate the effectiveness empirically, there is no theoretical analysis for the convergence and complexity. Furthermore, the models optimized by our approach lack interpretability, therefore utilizing interpretable models like decision trees[7, 10] is a possible avenue for future research. Additionally, beyond fitness-based selection, it may be worthwhile to explore alternative selection methods, such as Novelty Search [6, 22] or Quality Diversity methods [9, 31], which could be used to select diverse individuals and provide valuable learning experiences for the RL agent.

Acknowledgements

This work was supported in part by the National Natural Science Foundation of China under Grant 62203308 and 62173236, in part by the Guangdong Regional Joint Foundation Key Project under Grant 2022B1515120076, in part by the Natural Science Foundation of Guangdong Province under Grant 2023A1515011238, in part by the Shenzhen Science and Technology Program under Grants JCYJ20220531101411027.

CRediT authorship contribution statement

Xiaoqiang Wu: Conceptualization, Methodology, Software, Writing. **Qingling Zhu:** Conceptualization, Validation, Writing-reviewing and editing. **Wei-Neng Chen:** Supervision, Project administration. **Qiuzhen Lin:** Supervision, Validation, Writing-reviewing and editing, Project administration. **Jianqiang Li:** Supervision, Project administration. **Carlos A. Coello Coello:** Supervision, Project administration.

References

[1] ACBL. The acbl standard american yellow card system booklet. https://web2.acbl.org/documentlibrary/play/SP3 (bk) single pages.pdf.

[2] Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent complexity via multi-agent competition. In *International Conference on Learning Representations*, 2018.

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

[4] Noam Brown, Anton Bakhtin, Adam Lerer, and Qucheng Gong. Combining deep reinforcement learning and search for imperfect-information games. *Advances in Neural Information Processing Systems*, 33:17057–17069, 2020.

[5] CA Coello Coello. Evolutionary multi-objective optimization: a historical view of the field. *IEEE computational intelligence magazine*, 1(1):28–36, 2006.

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

[7] Vinícius G Costa, Jorge Pérez-Aracil, Sancho Salcedo-Sanz, and Carlos E Pedreira. Evolving interpretable decision trees for reinforcement learning. *Artificial Intelligence*, 327:104057, 2024.

[8] Marco Crespi, Andrea Ferigo, Leonardo Lucio Custode, and Giovanni Iacca. A population-based approach for multi-agent interpretable reinforcement learning. *Applied Soft Computing*, 147:110758, 2023.

[9] Antoine Cully and Yiannis Demiris. Quality and diversity optimization: A unifying modular framework. *IEEE Transactions on Evolutionary Computation*, 22(2):245–259, 2017.

[10] Leonardo L Custode and Giovanni Iacca. Evolutionary learning of interpretable decision trees. *IEEE Access*, 11:6169–6184, 2023.

[11] Leonardo Lucio Custode and Giovanni Iacca. A co-evolutionary approach to interpretable reinforcement learning in environments with continuous action spaces. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE, 2021.

[12] Leonardo Lucio Custode and Giovanni Iacca. Interpretable pipelines with evolutionary optimized modules for reinforcement learning tasks with visual inputs. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 224–227, 2022.

[13] Yashesh Dhebar, Kalyanmoy Deb, Subramanya Nageshrao, Ling Zhu, and Dimitar Filev. Toward interpretable-ai policies using evolutionary nonlinear decision trees for discrete-action systems. *IEEE Transactions on Cybernetics*, 2022.

[14] Weiping Ding, Mohamed Abdel-Basset, Hossam Hawash, and Ahmed M Ali. Explainability of artificial intelligence methods, applications and challenges: A comprehensive survey. *Information Sciences*, 615:238–292, 2022.

[15] Andrea Ferigo, Leonardo Lucio Custode, and Giovanni Iacca. Quality–diversity optimization of decision trees for interpretable reinforcement learning. *Neural Computing and Applications*, pages 1–12, 2023.

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

[17] Bo Haglund. Double dummy solver. https://github.com/dds-bridge/dds.

[18] Nikolaus Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[19] Jianye Hao, Pengyi Li, Hongyao Tang, Yan Zheng, Xian Fu, and Zhaopeng Meng. Erl-re^2: Efficient evolutionary reinforcement learning with shared state representation and individual policy representation. *International Conference on Learning Representations*, 2023.

[20] Shauharda Khadka, Somdeb Majumdar, Tarek Nassar, Zach Dwiel, Evren Tumer, Santiago Miret, Yinyin Liu, and Kagan Tumer. Collaborative evolutionary reinforcement learning. In *International conference on machine learning*, pages 3341–3350. PMLR, 2019.

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

[22] Joel Lehman, Kenneth O Stanley, et al. Exploiting open-endedness to solve problems through the search for novelty. In *ALIFE*, pages 329–336, 2008.

[23] Adam Lerer, Hengyuan Hu, Jakob Foerster, and Noam Brown. Improving policies via search in cooperative partially observable games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7187–7194, 2020.

[24] Junjie Li, Sotetsu Koyamada, Qiwei Ye, Guoqing Liu, Chao Wang, Ruihan Yang, Li Zhao, Tao Qin, Tie-Yan Liu, and Hsiao-Wuen Hon. Suphx: Mastering mahjong with deep reinforcement learning. *arXiv preprint arXiv:2003.13590*, 2020.

[25] Nan Li, Lianbo Ma, Guo Yu, Bing Xue, Mengjie Zhang, and Yaochu Jin. Survey on evolutionary deep learning: Principles, algorithms, applications, and open issues. *ACM Computing Surveys*, 56(2):1–34, 2023.

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

[27] Songbai Liu, Junhao Zheng, Qiuzhen Lin, and Kay Chen Tan. Evolutionary multi and many-objective optimization via clustering for environmental selection. *Information Sciences*, 578:930–949, 2021.

[28] Shuai Lü, Shuai Han, Wenbo Zhou, and Junwei Zhang. Recruitment-imitation mechanism for evolutionary reinforcement learning. *Information Sciences*, 553:172–188, 2021.

[29] Yan Ma, Tianxing Liu, Bingsheng Wei, Yi Liu, Kang Xu, and Wei Li. Evolutionary action selection for gradient-based policy learning. In *International Conference on Neural Information Processing*, pages 579–590. Springer, 2022.

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

[31] Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

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

[33] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. *Swarm intelligence*, 1(1):33–57, 2007.

[34] Aloïs Pourchot and Olivier Sigaud. Cem-rl: Combining evolutionary and gradient-based methods for policy search. In *7th International Conference on Learning Representations, ICLR*, 2019.

[35] Jiang Rong, Tao Qin, and Bo An. Competitive bridge bidding with deep neural networks. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 16–24, 2019.

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

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

[38] Lin Shao, Yifan You, Mengyuan Yan, Shenli Yuan, Qingyun Sun, and Jeannette Bohg. Grac: Self-guided and self-regularized actor-critic. In *Conference on Robot Learning*, pages 267–276. PMLR, 2022.

[39] Yuhui Shi and Russell C Eberhart. Empirical study of particle swarm optimization. In *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*, volume 3, pages 1945–1950. IEEE, 1999.

[40] Andrew Silva, Matthew Gombolay, Taylor Killian, Ivan Jimenez, and Sung-Hyun Son. Optimization methods for interpretable differentiable decision trees applied to reinforcement learning. In *International conference on artificial intelligence and statistics*, pages 1855–1865. PMLR, 2020.

[41] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[42] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[43] Riley Simmons-Edler, Ben Eisner, Eric Mitchell, Sebastian Seung, and Daniel Lee. Q-learning for continuous actions with cross-entropy guided policies. *arXiv preprint arXiv:1903.10605*, 2019.

[44] Karush Suri. Off-policy evolutionary reinforcement learning with maximum mutations. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, pages 1237–1245, 2022.

[45] Yuandong Tian, Qucheng Gong, and Yu Jiang. Joint policy search for multi-agent collaboration with imperfect information. *Advances in Neural Information Processing Systems*, 33:19931–19942, 2020.

[46] Yuxing Wang, Tiantian Zhang, Yongzhe Chang, Xueqian Wang, Bin Liang, and Bo Yuan. A surrogate-assisted controller for expensive evolutionary reinforcement learning. *Information Sciences*, 616:539–557, 2022.

[47] Yulong Ye, Qiuzhen Lin, Lijia Ma, Ka-Chun Wong, Maoguo Gong, and Carlos A Coello Coello. Multiple source transfer learning for dynamic multiobjective optimization. *Information Sciences*, 607:739–757, 2022.

[48] Chih-Kuan Yeh, Cheng-Yu Hsieh, and Hsuan-Tien Lin. Automatic bridge bidding using deep reinforcement learning. *IEEE Transactions on Games*, 10(4):365–377, 2018.

[49] Daochen Zha, Jingru Xie, Wenye Ma, Sheng Zhang, Xiangru Lian, Xia Hu, and Ji Liu. Douzero: Mastering doudizhu with self-play deep reinforcement learning. In *international conference on machine learning*, pages 12333–12344. PMLR, 2021.

[50] Xun Zhou, A Kai Qin, Maoguo Gong, and Kay Chen Tan. A survey on evolutionary construction of deep neural networks. *IEEE Transactions on Evolutionary Computation*, 25(5):894–912, 2021.