

# Deep Reinforcement Learning for Dynamic Algorithm Selection: A Proof-of-Principle Study on Differential Evolution

Hongshu Guo<sup>1</sup>, Yining Ma<sup>2</sup>, Zeyuan Ma<sup>3</sup>, Jiacheng Chen<sup>4</sup>, Xinglin Zhang<sup>5</sup>, *Member, IEEE*, Zhiguang Cao<sup>6</sup>, *Member, IEEE*, Jun Zhang<sup>7</sup>, *Fellow, IEEE*, and Yue-Jiao Gong<sup>8</sup>, *Senior Member, IEEE*

**Abstract**—Evolutionary algorithms, such as differential evolution, excel in solving real-parameter optimization challenges. However, the effectiveness of a single algorithm varies across different problem instances, necessitating considerable efforts in algorithm selection or configuration. This article aims to address the limitation by leveraging the complementary strengths of a group of algorithms and dynamically scheduling them throughout the optimization progress for specific problems. We propose a deep reinforcement learning-based dynamic algorithm selection framework to accomplish this task. Our approach models the dynamic algorithm selection a Markov decision process, training an agent in a policy gradient manner to select the most suitable algorithm according to the features observed during the optimization process. To empower the agent with the necessary information, our framework incorporates a thoughtful design of landscape and algorithmic features. Meanwhile, we employ a sophisticated deep neural network model to infer the optimal action, ensuring informed algorithm selections. Additionally, an algorithm context restoration mechanism is embedded to facilitate smooth switching among different algorithms. These mechanisms together enable our framework to seamlessly select and switch algorithms in a dynamic online fashion. Notably, the proposed framework is simple and generic, offering potential improvements across a broad spectrum of evolutionary algorithms. As a proof-of-principle study, we apply this framework to a group of differential evolution algorithms. The experimental results showcase the remarkable effectiveness of the proposed framework, not only enhancing

the overall optimization performance but also demonstrating favorable generalization ability across different problem classes.

**Index Terms**—Algorithm selection, black-box optimization, deep reinforcement learning, differential evolution, meta-black-box optimization.

## I. INTRODUCTION

EVOLUTIONARY computation (EC) has experienced considerable development in the past few decades [1]. Many EC paradigms, such as genetic algorithm (GA) [2], differential evolution (DE) [3], particle swarm optimization (PSO) [4], and their variants, have exhibited good performance in solving black-box optimization (BBO) problems, owing to their outstanding global exploration and local exploitation abilities. However, it has long been observed that for different benchmark or practical problems, when several ECs are available, no single one can dominate on all problem instances, known as the “no-free-lunch” (NFL) theorem [5].

To address this issue, an intuitive idea is to leverage the performance complementarity among a group of different EC algorithms, or different configurations for a single EC algorithm, which can serve as different specialties for respective problem instance. This attracted many research efforts to investigate which algorithm or algorithm configuration (AC) could be promising in dealing with a particular problem instance [6]. We have carefully sorted out the related studies and roughly divided them into three categories: 1) algorithm portfolio (AP); 2) AC; and 3) algorithm selection (AS).

AP, also known as *parallel AP*, runs all candidates from a given algorithm set in parallel to determine the most outstanding one for tackling a given problem instance. It was first proposed by Huberman et al. [7] and then improved by Gomes and Selman [8] on the necessary conditions when the parallel portfolio outperforms its component solvers. Although AP algorithms benefit from the development of parallel hardware, their huge computation requests are inevitable.

AC refers to finding a proper configuration of an algorithm in order to maximize its performance across a set of problem instances. Classical AC algorithms target an united parameter configuration for all problem instances. For example, Ansótegui et al. [9] took advantage of the iterated searching algorithms to search proper parameter

Manuscript received 4 December 2023; accepted 29 February 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 62276100; in part by the Guangdong Natural Science Funds for Distinguished Young Scholars under Grant 2022B1515020049; in part by the Guangdong Regional Joint Funds for Basic and Applied Research under Grant 2021B1515120078; and in part by the TCL Young Scholars Program. This article was recommended by Associate Editor J. Liang. (Corresponding author: Yue-Jiao Gong.)

Hongshu Guo, Zeyuan Ma, Jiacheng Chen, Xinglin Zhang, and Yue-Jiao Gong are with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China (e-mail: gongyuejiao@gmail.com).

Yining Ma is with the Department of Industrial Systems Engineering and Management, College of Design and Engineering, National University of Singapore, Singapore (e-mail: yiningma@u.nus.edu).

Zhiguang Cao is with the School of Computing and Information Systems, Singapore Management University, Singapore (e-mail: zhiguangcao@outlook.com).

Jun Zhang is with Nankai University, Tianjin 300192, China, also with Hanyang University, Seoul 04763, South Korea, and also with the University of Victoria, Melbourne, VIC 3004, Australia (e-mail: junzhang@ieee.org).

This article has supplementary material provided by the authors and color versions of one or more figures available at <https://doi.org/10.1109/TSMC.2024.3374889>.

Digital Object Identifier 10.1109/TSMC.2024.3374889

settings in the configuration space. Recently, the AC paradigms dynamically adjusted the algorithmic settings not only for each problem instance but also at each time step during optimization. Typically, the reinforcement learning (RL) methods were adopted to learn an online policy for parameter control, such as in the studies of Xue et al. [10] and Biedenkapp et al. [11]. Generally, AC provides a flexible way to configure an efficient algorithm to different problem instances. But since the configuration space of AC could be extremely large by containing infinite sets of algorithmic components, so far the configuration efficiency remains a challenge [6].

One way to relieve the above issue of AC is to reduce the configuration space by using expert knowledge to predetermine a finite set of parameter and operator combinations with guaranteed performance. This is typically known as the AS, which targets at selecting the most promising expert-designed algorithm for a given problem. Existing AS on real-parameter optimization tasks, such as those proposed by Bischl et al. [12] and Kerschke et al. [6], both probed the problem with a few samplings and thereafter calculated informative landscape features. These features then served as input for a machine learning model that decided which algorithm should be applied to the given problem.

Nonetheless, existing AS works still face the following challenges. First, some existing studies focus on static AS that, once an algorithm is selected, it will be executed for the whole optimization process. Such methods neglect the utilization of the performance complementary among candidates to achieve comprehensively better performance than the best single algorithm in the candidate set. Second, existing AS studies adopt supervised learning methods to estimate candidates' performance based on analytical features and take the estimation results as the basis for AS. However, without modeling the optimization progress in detail, the available features for characterizing the problem landscape and/or algorithm behavior are rather limited.

To address the above issues, this article proposes a deep reinforcement learning based dynamic algorithm selection (RL-DAS) framework, presenting following key contributions.

- 1) We extend the AS to a dynamic AS (DAS) setting that supports dynamically scheduling different algorithms upon request throughout the optimization process. As illustrated in Fig. 1, the optimization process is divided into time intervals. At each interval, an algorithm is dynamically selected to improve the population, taking into account the problem state and the performance of previously selected algorithms. This ensures that the overall optimization outcome is maximized.
- 2) Since our objective is to find the best sequential selections for different problems, we formulate the DAS as a Markov decision process (MDP) for sequential decision making. We then leverage the deep RL (DRL) method to solve the formulated MDP. Meanwhile, RL-DAS supports the *warm start* when switching between different algorithms by maintaining an algorithm context memory and applying an algorithm restoration step, ensuring a smooth switching process.

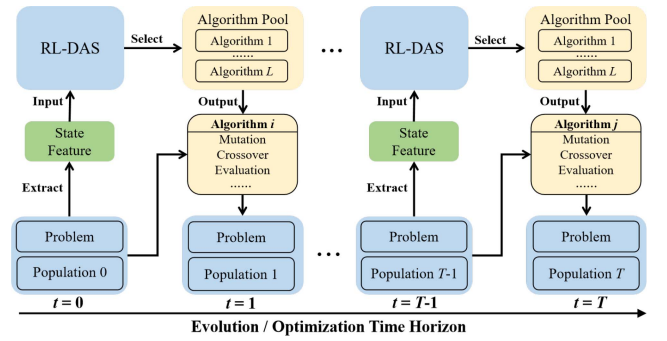


Fig. 1. Overall structure of RL-DAS.

TABLE I  
ABBREVIATION INDICES

Abbr	Note	Abbr	Note
EC	Evolutionary Computation	DAS	Dynamic Algorithm Selection
BBO	Black-Box Optimization	RL	Reinforcement Learning
DE	Differential Evolution	DRL	Deep Reinforcement Learning
PSO	Particle Swarm Optimization	MDP	Markov Decision Process
AP	Algorithm Portfolio	PPO	Proximal Policy Optimization
AC	Algorithm Configuration	LA	Landscape Analysis information
AS	Algorithm Selection	AH	Algorithm History information

- 3) Furthermore, in order to fully inform the DRL agent and guide it to make wise decisions, we design a set of simple yet representative features to capture both landscape and algorithmic properties. A deep neural network model is then carefully designed to infer the best action based on the derived features. Our designs allow RL-DAS to generalize to various BBO problems.
- 4) We apply RL-DAS on the augmented CEC2021 competition benchmark, with a number of DE algorithms as candidates. The experimental results show that RL-DAS not only outperforms all candidates in the algorithm pool but also exhibits promising zero-shot generalization on unseen problems.

The remainder of this article is organized as follows. Section II presents the preliminaries on DRL and DE. Section III introduces the detail of RL-DAS. Section IV is the proof-of-principle study of applying RL-DAS on advanced DE algorithms. The experimental results are depicted in Section V with detailed analysis, followed by a conclusion in Section VI.

## II. PRELIMINARIES AND RELATED WORKS

In this section, we briefly introduce the MDP and the advanced variants of DE. Table I summarizes the abbreviation indices for easy reference.

### A. Markov Decision Process and Policy Gradient

An MDP could be denoted as  $\mathcal{M} := \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, R \rangle$ . Given a state  $s_t \in \mathcal{S}$  at time step  $t$ , an action  $a_t \in \mathcal{A}$  is selected and interacts with the environment, where the next state  $s_{t+1}$  is produced through the environment dynamic  $\mathcal{T}(s_{t+1}|s_t, a_t)$ . A reward function  $R: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  is calculated to measure the performance improvement in the transition. Those transitions between states and actions achieve a trajectory  $\tau := (s_0, a_0, s_1, \dots, s_T)$ . The target of MDP is to

find an optimal policy  $\pi^*$  such that the returns (accumulated rewards) in trajectories  $\tau$  can be maximized

$$\pi^* = \arg \max_{\pi \in \Pi} \sum_{t=0}^T \gamma^{t-1} R(s_t, a_t) \quad (1)$$

where  $\Pi : \mathcal{S} \rightarrow \mathcal{A}$  selects an action with a given state,  $\gamma$  is a discount factor, and  $T$  is the length of trajectory. One method specialized in solving MDPs is the *Policy Gradient*-based DRL, which parameterizes the policy by a deep model  $\pi_\theta$  with parameters  $\theta$ . The policy is expected to decide on a proper action at each state in order to achieve a high return of the trajectory. The objective function is formulated as

$$J(\theta) := \mathbb{E}_{\tau \sim p_\theta(\tau)} [R(\tau)] \quad (2)$$

which is maximized by the gradient ascent with gradient

$$\frac{\partial J}{\partial \theta} = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[ \left( \sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_t | s_t) \right) \left( \sum_{t=0}^T R(s_t, a_t) \right) \right]. \quad (3)$$

In this article, we adopt a powerful Policy Gradient variant, the proximal policy optimization (PPO) [13], to solve the MDP. It proposes a novel objective with clipped probability ratios, which forms a first-order estimate (i.e., lower bound) of the policy's performance. The algorithm alternates between sampling data from the policy and performing several epochs of optimization on the sampled data to improve data efficiency and ensure reliable performance. Its objective function is

$$L_\pi(\theta) := \mathbb{E} \left[ \min(\eta(\theta)\hat{A}, \text{clip}(\eta(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}) \right] \quad (4)$$

where  $\eta$  is the ratio of the probabilities under the new and old policies which performs the importance sampling, and  $\hat{A}$  is the estimated advantage calculated as the difference between the target return  $G$  and the estimated return  $\hat{G}$ . Note that PPO is trained in an actor-critic manner that a critic network  $v_\phi$  is adopted to estimate the expected return  $\hat{G}$  for different states. The loss function of  $v_\phi$  is formulated as

$$L_v(\phi) := \text{MSE}(G, \hat{G}). \quad (5)$$

## B. DE and Its Advanced Variants

As mentioned earlier, this article uses DE as a case study for our RL-DAS framework. Therefore, in this section, we review some important DE variants. Generally, DE is a high-profile field in EC since proposed by Storn and Price [3]. The DE variants are frequent participants and winners in major BBO competitions [14] and widely applied in various practical applications [15], [16], [17]. Considering the relevance of DE variants to our study, we categorize them into three distinct groups for review.

1) *Adaptive DE*: In the literature, there have emerged different DE mutation forms, such as DE/rand/1 and DE/best/1. According to the NFL theorem, no single operator can fit all problem instances, creating room for operator adaptation. EPSDE [18] assigned each individual an operator which would be randomly reselected if it failed to improve the individual. LADE [19] divided the population into leader population and adjoint population in each generation and employed different

mutation operators on them for different searching strategies. Parameters, such as the step length  $F$  of mutation and the crossover rate  $Cr$ , also play a significant role in optimization. For instance, jDE [20] randomly generated these parameters and remembered those that successfully improved individuals for later generations. JDE21 [21] made further improvements on jDE which split the population into two parts to perform different search strategies. SHADE [22] adopted two memories for  $F$  and  $Cr$  to record their statistical information instead of remembering their values. Its variants, NL-SHADE-RSP [23] employed an adaptive mutation operator with an archive of eliminated individuals to increase the population diversity, NL-SHADE-LBC [24] further incorporated a modified bound constraint handling technique and improved the performance. MadDE [25] adopted the parameter memories proposed in SHADE and employed three adaptive mutation operators and two crossover operators together with the archive design. The JDE21, MadDE, and NL-SHADE-RSP are winners of the CEC2021 competition. However, the adaption mechanisms rely heavily on designers' expert knowledge and introduce additional parameters (such as the sizes of memories [24] and the probabilities of selecting operators [25]). These new mechanisms and parameters substantially improve the complexity of algorithm fine-tuning, making the algorithms susceptible to overfitting on specific optimization problems.

2) *Ensemble DE*: Since different DE variants possess strengths in specific types of optimization problems, several studies directly combined multiple DE variants into an ensemble to enhance overall optimization performance. This approach has resulted in the development of Ensemble DE, which bears high relevance to our study. For instance, EDEV [26] proposed a multipopulation-based framework integrating JADE [27], CoDE, and EPSDE, where the best-performing one on a small indicator subpopulation was assigned a large reward population for a period. In contrast to multipopulation methods, HMJCDE [28] used a hybrid framework with modified JADE and modified CoDE on a single population, operating them alternately according to the fitness improvement rate. While existing Ensemble DE algorithms have consistently outperformed their baselines, showcasing impressive performance, their AS mechanisms are also manually designed and hence suffer from similar drawbacks of Adaptive DE. Additionally, they often choose candidates based on the historical performance [26], [28]. However, for different optimization stages, the best-fitted algorithm differs. An exploratory algorithm might excel in the early optimization phase but could be less advantageous for convergence later on. The selection process could benefit from more nuanced approaches, such as using a machine learning model to learn the relation between optimization states and AS.

3) *RL-Assisted DE*: While adaptive and ensemble DE algorithms have shown significant progress, the dependence on manual mechanisms and expert knowledge could potentially hinder their further development. Consequently, some researchers have turned to machine learning methods, particularly RL techniques, in search of breakthroughs [29].



Research in this field primarily concentrates on DE configuration, including the selection of mutation and crossover operators, as well as the values of  $F$  and  $Cr$ . For operator selection, TSRL-DE [30], RLDMDE [31], DEDQN [32], and DEDDQN [33] employed  $Q$ -Learning [34] and Deep  $Q$ -Learning agents [35], [36] to select mutation operators for each individual in every generation and promote the backbone algorithm performance on several CEC benchmark problems. Regarding parameter tuning, LDE [37] utilized LSTM [38] to determine the  $F$  and  $Cr$  values for each individual, based on the states consisting of fitness values and statistics. Furthermore, some researchers sought more comprehensive control by combining both operator and parameter control, such as the RLHPSDE [39]. Current RL-Assisted DE algorithms primarily concentrate on adapting configurations within individual DE algorithms. Given there are plenty of well-performed DE algorithms for different problems, it would be appealing to adopt the RL agents for DAS, which is, however, a research gap so far. Additionally, the current RL-assisted works arise from the small size of training and testing sets [32], [33], [37], which could adversely impact their generalization ability.

### III. METHODOLOGY

#### A. Dynamic Algorithm Selection

Given a set of algorithms  $\Lambda = \{\Lambda_1, \Lambda_2, \dots, \Lambda_L\}$  and a set of problem instances  $I = \{I_1, I_2, \dots, I_M\}$ , the objective of traditional AS is to find an optimal policy as follows:

$$\pi^* = \arg \max_{\pi \in \Pi} \sum_{j=1}^M \mathcal{M}(\pi(I_j), I_j) \quad (6)$$

where  $\pi : I \rightarrow \Lambda$  is a selector that selects an algorithm  $\Lambda_i \in \Lambda$  for instance  $I_j \in I$  to maximize the overall performance metric  $\mathcal{M} : \Lambda \times I \rightarrow \mathbb{R}$ .

Pertaining to the DAS setting proposed in this article, time steps are introduced and the objective becomes seeking

$$\pi^* = \arg \max_{\pi \in \Pi} \sum_{j=1}^M \sum_{t=1}^T \mathcal{M}(\pi(I_j, t), I_j) \quad (7)$$

where  $\pi : I \times T \rightarrow \Lambda$  indicates that the selector should choose a proper algorithm at each optimization interval for each instance. Note that if we 1) define a reward function based on the metric  $\mathcal{M}$ ; 2) consider choosing candidate algorithms at different time steps as actions; and 3) formulate the optimization state at each time step as states, the DAS problem can be regarded as an MDP, which could be solved by RL methods such as the PPO [13].

#### B. RL-DAS Overview

According to the definition of DAS, the environment of DAS includes an algorithm pool, a problem set, and a population for optimization. To formulate the MDP, we design the *state* to contain sufficient information through analysis of the fitness landscape and optimization status, the *action* to select one algorithm from the algorithm pool, and the *reward* to reflect

#### Algorithm 1: Pseudocode of RL-DAS

---

**Input:** Policy  $\pi_\theta$ , Critic  $v_\phi$ , Instance Set  $I$ , Algorithm Pool  $\Lambda$  with Context Memory  $\Gamma$

**Output:** Trained Policy  $\pi_\theta$ , Critic  $v_\phi$

**for**  $epoch \leftarrow 0$  **to**  $Epoch$  **do**

**for**  $instance \in I$  **do**

    Population  $P_0 \leftarrow \text{InitPopulation}()$ ;

    Initialize  $\Gamma$ ;

$t \leftarrow 0$ ;

**while** the termination condition is not met **do**

$s_t \leftarrow \text{State}(P_t, instance, \Lambda)$ ;

$a_t, \log \pi_\theta(a_t|s_t) \leftarrow \pi_\theta(a_t|s_t)$ ;

$algorithm \leftarrow \Lambda[a_t]$ ;

$algorithm \leftarrow \text{Restoration}(\Gamma, algorithm)$ ;

$P_{t+1} \leftarrow \text{Step}(algorithm, P_t, instance)$ ;

$s_{t+1} \leftarrow \text{State}(P_{t+1}, instance, \Lambda)$ ;

      Get reward  $adc_t$  by Eq. (12);

      Get  $\langle s_t, a_t, s_{t+1}, adc_t, \log \pi_\theta(a_t|s_t) \rangle$ ;

$t \leftarrow t + 1$ ;

      Update  $\Gamma$ ;

**end**

    Update  $adc_{0:T}$  into  $r_{0:T}$  by Eq. (13);

**for**  $k \leftarrow 1$  **to**  $K$  **do**

      Update  $\pi_\theta$  and  $v_\phi$  by PPO method;

**end**

**end**

**end**

---

the improvement of metric  $\mathcal{M}$ . The workflow of RL-DAS is presented in Algorithm 1. The training on each problem instance starts with an initial population  $P_0$  and terminates when either the maximum number of function evaluations  $MaxFEs$  is exhausted or the best cost so far in the population is lower than the termination error. In the  $t$ -th step, information from the population  $P_t$ , the optimization problem and the algorithm pool  $\Lambda$  is extracted and constructed as a state  $s_t$ . Based on the state, the policy agent  $\pi_\theta$  decides an action  $a_t$  that selects the next algorithm to switch. Additionally, we calculate the log likelihood of the action,  $\log \pi_\theta(a_t|s_t)$  which is used in PPO. With parameters inherited from a context memory  $\Gamma$ , an algorithm is warm-started from its last break and applied to involve the population for a period of time (until the next decision step). After the period, the reward  $r_t$  is observed, and a new population  $P_{t+1}$  is produced, generating the new state information  $s_{t+1}$ . The updated contextual information of the algorithm is appended to the memory  $\Gamma$  to prepare for the next restoration. The trajectories of states, actions, and rewards are recorded and then used by the PPO method to train the policy net  $\pi_\theta$  and the critic net  $v_\phi$ . To fully utilize the recorded trajectories and promote the training efficiency, PPO updates the networks  $K$  times after the completion of optimization. A larger  $K$  leads to slower training and an elevated risk of overfitting on the current problem, whereas a smaller  $K$  may restrict the utilization and the overall efficiency. In this article, we set  $K = 0.3 \times T$ .

In the following of this section, we introduce the design of MDP components, including the state, action, and reward in Section III-C. The neural network design and the training process will be presented in Section III-D. Finally, we introduce the aforementioned algorithm context memory mechanism to support the warm-start of different algorithms in Section III-E.

### C. MDP Components

This section presents a detailed design of the MDP, including state, action, and reward, and explains how they enable the interaction between the environment and the DRL agent.

1) *State*: The state space of RL-DAS consists of landscape analysis (LA) and algorithm history (AH) information. For the LA part, nine scalar features ( $feature_1, \dots, feature_9$ ) are extracted by analyzing the population information and the random walk sampling information on the problem landscape, which forms an LA vector  $f_{LA} \in \mathbb{R}^9$ . The sampling information probes the fitness landscape, revealing the complexity and difficulty of the optimization problem. These features also reflect the interaction between the current population and the optimization problem. Below, we briefly introduce the LA features ( $feature_1, \dots, feature_9$ ), deferring the detailed formulations in Section 1 of our supplementary document.

- 1)  $feature_1$ : The cost value is a pivotal indicator of optimization status, making it our first feature.
- 2)  $feature_2$ : The fitness distance correlation (FDC) [40] describes problem complexity by assessing the relationship between fitness values and solution distances.
- 3)  $feature_3$ : The dispersion difference [41] measures the distribution difference between the top 10% and the whole population. It is applied to analyze the funnelity of the problem landscape: a single funnel problem has a smaller dispersion difference value as the top 10% individuals gather around the optimal point and have smaller inner distance, otherwise for the multifunnel landscape, this value would be much larger.
- 4)  $feature_4$ : The maximum distance among population describes the convergence state of population.
- 5)  $feature_{5,6,7,8}$ : These features gauge the evolvability of the population on the given problem. First, the negative slope coefficient (NSC) [42] measures the difference between the cost change incurred by the population evolution and by making small tentative random walk steps. The average neutral ratio (ANR) [43] has a similar consideration of NSC, but it measures the performance changes between the current population and different sampled populations. Inspired by Wang et al. [44], we also propose the Best–Worst Improvement to measure the evolvability. The Best Improvement feature measures the optimization difficulty of the current population using the ratio of sampled individuals that make no improvement over the current population, while the Worst Improvement feature describes the optimization potential by the ratio of individuals that do not get worse.
- 6)  $feature_9$ : To keep the agent informed about computational budget consumption, we introduce the ratio of consumed FEs over the maximum as a feature.

Note that the function evaluations costed in the above process for state representation should be included in evaluating the termination condition.

As our environment also contains the algorithm pool, to provide the RL agent an additional contextual knowledge about the optimization ability of the candidate algorithms, we derive the AH information to characterize the optimization status from an algorithmic perspective. Suppose that there are  $L$  algorithms in  $\Lambda$ . We maintain two shift vectors for each algorithm to characterize the population distribution change incurred by it. Specifically, let  $SV_{best}^l \in \mathbb{R}^D$  and  $SV_{worst}^l \in \mathbb{R}^D$  denote the average shift of the best and the worst individual caused by the algorithm  $\Lambda^l$  in the previous optimization steps. For a time interval  $i$ , if  $\Lambda^l$  is selected to execute, it transforms the population  $P$  with the best individual  $x_{best}$  to a new population  $P'$  with  $x'_{best}$ . Then, the shift vector of the best individual at the  $i$ th step,  $sv_{i,best}^l \in \mathbb{R}^D$ , is defined as

$$sv_{i,best}^l = x'_{best} - x_{best} \quad (8)$$

which presents the effect of  $\Lambda^l$  in this step. Then, the  $SV_{best}^l$  is formulated as

$$SV_{best}^l = \frac{1}{H^l} \sum_{i=1}^{H^l} sv_{i,best}^l \quad (9)$$

where  $H^l$  is the number of intervals that the algorithm  $\Lambda^l$  is selected. The calculation of  $SV_{worst}^l$  is similar. Notably, in the case that the algorithm  $\Lambda^l$  is never selected by the RL agent ( $H^l = 0$ ), its  $SV_{best}^l$  and  $SV_{worst}^l$  are zero vectors.

Based on the above,  $f_{AH}$  is a collection of the  $L \times 2$  shifting vectors of all candidate algorithm, which captures the behavior histories of the candidate algorithms and helps RL agent learn about the characteristics of candidate algorithms

$$f_{AH} = \left[ \left\{ SV_{best}^1, SV_{worst}^1 \right\} \dots \left\{ SV_{best}^L, SV_{worst}^L \right\} \right]. \quad (10)$$

Finally, the complete state in the MDP of RL-DAS is the integration of the above LA and AH information

$$state = \{f_{LA} \in \mathbb{R}^9, f_{AH} \in \mathbb{R}^{2L \times D}\}. \quad (11)$$

An intuitive explanation of this state design is to provide the RL agent with awareness of both the problem instance characteristics and the historical performance of algorithms. This enables the RL agent to make more informed decisions when selecting algorithms during the optimization process.

2) *Action*: Since the motivation behind RL-DAS is to periodically switch optimization algorithms to achieve complementary performance, the action is designed as an integer number indicating the index of the selected algorithm in a pool containing  $L$  candidate algorithms, denoted as  $a \in [1, L]$ .

3) *Reward*: In order to guide the agent toward achieving a lower evaluation cost, the reward function should consider the absolute cost reduction at each time step  $t$

$$adc_t = \frac{cost_{t-1}^* - cost_t^*}{cost_0^*} \quad (12)$$

where  $cost_t^*$  and  $cost_0^*$  are the best cost until the  $t$ -th step and the best cost in the initial population, respectively

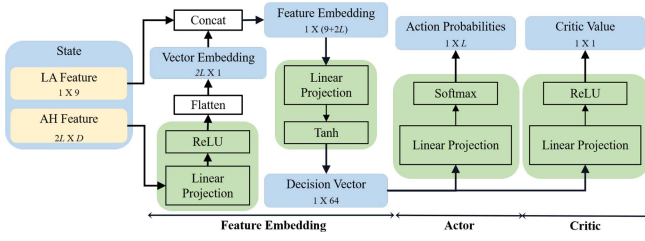


Fig. 2. Neural network workflow for  $\pi_\theta$  (Actor) and  $v_\phi$  (Critic).

(where  $cost_0^*$  serves as a normalization factor). This measures the performance improvement brought in the  $t$ -th step optimization.

In addition, when employing EC algorithms for optimization, we usually pay attention not only to the descent of cost but also to the speed of convergence. The latter can be measured by the number of consumed FEs to the termination condition. Therefore, our reward function is defined as an adapted absolute descent of cost, by introducing the measurement of optimization speed

$$r_t = \frac{cost_{t-1}^* - cost_t^*}{cost_0^*} \times \frac{MaxFEs}{FEs_{end}} \quad (13)$$

where  $FEs_{end}$  is the number of consumed FEs in reaching the termination condition. The agent is encouraged to search for an optimal strategy capable of finding the solution with the lowest cost, while making the most efficient use of resources.

#### D. Network Design

As shown in Fig. 2, the network is composed of three modules: 1) Feature Embedding; 2) Actor; and 3) Critic. The AH feature and the LA feature are first fused to form the state representation vector. Based on the representation, the Actor outputs the probability distribution over the candidate algorithms, and the Critic estimates a return value. In the implementation of RL-DAS, we feed the network a batch of problems simultaneously to enhance the convergence of training. In the subsequent description, we omit the time step subscript  $t$  and the batch dimensions for better readability.

1) *Feature Embedding*: The Feature Embedding module fuses  $f_{LA}$  and  $f_{AH}$ . First, we feed  $f_{AH}$  into a two-layer feedforward network to generate a more compact embedding vector  $V_{AH}$  as

$$V_{AH} = \sigma(W_{ve}^{(2)} \sigma(W_{ve}^{(1)} f_{AH} + b_{ve}^{(1)}) + b_{ve}^{(2)}) \quad (14)$$

where  $W_{ve}^{(1)} \in \mathbb{R}^{D \times 64}$ ,  $W_{ve}^{(2)} \in \mathbb{R}^{64 \times 1}$  are two network layers and the activation function  $\sigma$  is ReLU. Through the two layers, the AH information is embedded in a  $2L$  compact vector, whose length is in the similar magnitude of the  $9D$   $f_{LA}$ . The two types of features are concatenated and then fused by another network layer to generate the decision vector representation as

$$DV = \text{Tanh}(W_{dv}(f_{LA} \oplus \text{Flatten}(V_{AH})) + b_{dv}) \quad (15)$$

where the weight for the decision vector  $W_{dv} \in \mathbb{R}^{(9+2L) \times 64}$ , and the activation function  $\text{Tanh}$  is applied to limit the possible extreme values in the features.

2) *Actor*: Actor decides the probability for selecting an algorithm from the candidate algorithm pool. It first maps  $DV$  in (15) to a logits vector through a two-layer feedforward network. Then, after the Softmax operation, the actor outputs the probability distribution on the candidate algorithm pool, which are then used to sample an algorithm

$$\pi = \text{Softmax}\left(W_{actor}^{(2)} \text{Tanh}\left(W_{actor}^{(1)} DV + b_{actor}^{(1)}\right) + b_{actor}^{(2)}\right) \quad (16)$$

where  $W_{actor}^{(1)} \in \mathbb{R}^{64 \times 16}$  and  $W_{actor}^{(2)} \in \mathbb{R}^{16 \times L}$ . The policy  $\pi$  indicates the probabilities of each algorithm to be selected.

3) *Critic*: Critic feeds  $DV$  into a two-layer feedforward network to obtain the estimated return value of the state

$$b = \sigma\left(W_{critic}^{(2)} \sigma\left(DV^\top W_{critic}^{(1)} + b_{critic}^{(1)}\right) + b_{critic}^{(2)}\right) \quad (17)$$

where  $W_{critic}^{(1)} \in \mathbb{R}^{D \times 64}$  and  $W_{critic}^{(2)} \in \mathbb{R}^{64 \times 1}$  are two network layers, and the activation function  $\sigma$  in both layers is ReLU.

#### E. Algorithm Context Memory

Modern EC algorithms usually contain some contextual information, such as the dynamically determined parameters, some statistical measures or indicators, and other configurations during the optimization. For single algorithm execution, such information is smoothly updated along the optimization progress. However, the DAS studied in this article may break the smoothness of updating the above information when switching algorithms since different algorithms consider different contextual information. This results in a *warm up* issue, i.e., how to properly manage the context information of different algorithms and then perform context restoration for a smooth search. To address this issue, we first separate the algorithm-specific contexts from the common contexts among the candidate algorithms. Then, we design a nested dictionary-like context memory to manage them

$$\Gamma := \begin{cases} \Lambda_1 : \{\text{ctx}_{1,1}, \text{ctx}_{1,2}, \dots\} \\ \vdots \\ \Lambda_L : \{\text{ctx}_{L,1}, \text{ctx}_{L,2}, \dots\} \\ \text{Common} : \{\text{ctx}_1, \text{ctx}_2, \dots\}. \end{cases} \quad (18)$$

Here,  $\text{ctx}_{i,*}$  indicates contexts for the  $i$ th algorithm, which is updated when the corresponding algorithm is selected for execution. The *Common* dictionary contains some common contexts among the candidate algorithms, which can be updated by multiple candidates. When an algorithm  $\Lambda_i$  needs to be restored, its dictionary  $\Gamma[\Lambda_i]$  is indexed, and the contexts inside  $\Gamma[\Lambda_i]$  and *Common* are restored, in order to support the warm-start of  $\Lambda_i$ . This restoration method can be generalized to various algorithms, as it does not propose any special requirements on the contexts.

#### IV. RL-DAS ON DE: PROOF-OF-PRINCIPLE STUDY

The design of RL-DAS can be applied to various EC algorithms, including GA, PSO, DE, and their mixtures, each with corresponding algorithm pools. However, in this article, for the sake of convenience in the comparison and discussion, we focus on presenting a proof-of-principle study using



representative DE algorithms. As mentioned in Section II-B, DE is a high-profile field in EC. Although exhibits powerful performance in continuous optimization, different DE variants generally exhibit different performances on different problem instances. We realize the above RL-DAS framework to dynamically schedule three DE algorithms along the optimization process, which is detailed in this section.

#### A. Candidate Algorithm Pool

We adopt three advanced DE algorithms from the CEC 2021 competition [14], namely, JDE21 [21], MadDE [25], and NL-SHADE-RSP [23]. These three algorithms have been introduced in Section II-B, which have different strengths and weaknesses in terms of balancing the exploration and exploitation of the population. According to our experiments (which will be discussed in Section 3 of our supplementary document), these three algorithms present different search characteristics and exhibit complementary performance in different optimization stages of different problem instances, which leaves room for our RL-DAS agent to learn a desired collaboration between them.

#### B. Schedule Interval

The interval between schedules plays a significant role in the training efficiency of DRL agents and the optimization performance of EC. On the one hand, a short interval allows the agent to switch among candidates more frequently and accumulate more transitions, helps the DRL agent to learn faster. However, on the other hand, frequent changes can break the integrity of algorithms, resulting in poor optimization performance because the algorithm can be switched before making any substantial improvement. A long interval allows algorithms to explore the search space sufficiently, but the agent learns less, and the flexibility of DAS is also decreased. An extreme case is that when the interval is as long as the complete optimization budget, the DAS degrades to the traditional AS. Therefore, determining the interval requires balancing the optimization performance and the training efficiency.

According to the RL practice, the common length of an RL trajectory ranges from  $T_{\min} = 10$  to  $T_{\max} = 200$  [45]. To this end, considering the optimization length in EC, which is usually controlled by the  $MaxFEs$ , the suggested schedule interval of RL-DAS is  $\Delta \in [(MaxFEs/T_{\max}), (MaxFEs/T_{\min})]$ . In essence, at every  $\Delta$  FEs, the framework attempts to insert a transition (algorithm switching) step of RL. In practical execution, the framework will wait until the current generation of the DE algorithm completed before starting the transition step. As a result, the real scheduling interval,  $\tilde{\Delta}$ , may be equal to or slightly larger than the planned  $\Delta$ . In Section V-E, we will conduct a grid experiment to investigate interval lengths that support good performance with acceptable training efficiency.

#### C. Algorithm Restoration

The three algorithms in our candidate pool adopt many adaptive methods, which makes restoring these contexts a challenge. For JDE21, the contexts contain parameters, such

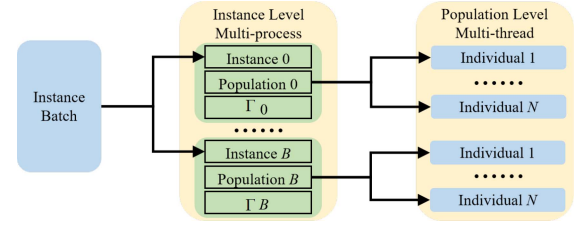


Fig. 3. Structure of the hierarchical parallel mechanism.  $B$  indicates the batch size.

as the probabilities of adaptive mutation and crossover ( $\tau_1$  and  $\tau_2$ ), the parameters determine the count of stagnation ( $ageLmt$ ,  $eps$ ) and the controller of reinitialization ( $myEqs$ ). MadDE adopts multiple operators, whose contexts involve the mutation probabilities ( $p_m$  and  $p_{best}$ ) and the crossover probability ( $p_{qBX}$ ). In NL-SHADE-RSP, the probability of using archive ( $p_A$ ) and its adaption parameter ( $n_A$ ) are included. The successful parameters ( $S_F$  and  $S_{Cr}$ ) for JDE21 and the parameter memories ( $M_F$  and  $M_{Cr}$ ) for MadDE and NL-SHADE-RSP, as well as their elite archive ( $Arch$ ) are stored in the *Common*. To summarize, the complete context memory  $\Gamma$  for the three algorithms is organized in the following form:

$$\Gamma := \begin{cases} \text{JDE21} : \{\tau_1, \tau_2, ageLmt, eps, myEqs\} \\ \text{MadDE} : \{p_m, p_{best}, p_{qBX}\} \\ \text{NL-SHADE-RSP} : \{n_A, p_A\} \\ \text{Common} : \{S_F, S_{Cr}, M_F, M_{Cr}, Arch\}. \end{cases} \quad (19)$$

For example, when MadDE is selected, the algorithm finds the linked dictionary and uses  $p_m$ , which is updated and frozen since last time MadDE being selected, to sample mutation operators. At the end of each generation, the probabilities are updated according to the performance of mutations. When MadDE is switched (e.g., to use JDE21 or NL-SHADE-RSP), the adaptive contexts in  $\Gamma$  are replaced with those updated  $p_m$ .

#### D. Parallelization

In RL-DAS, the training of the DRL agent is based on the sampled transitions from a period of EC iterations. Usually the agent requires a lot of sampled transitions to converge, which would consume much computational time. To address this low-training efficiency problem, as shown in Fig. 3, we adopt a hierarchical parallel mechanism that accelerates the sampling at both the instance level and the population level.

Pertaining to the instance level, we use a parallel method to sample transitions from a problem instance batch. First, a batch of problem instances is sampled from the problem suite. Accordingly, we generate the same number of populations and algorithm architectures. These instances, populations, and algorithms are then run in parallel with subprocesses. The agent processes their features in batch and returns the actions to their corresponding subprocesses, which determines the next algorithm to use on each instance. Note that the populations and the algorithm context memories are not shared among subprocesses. Pertaining to the population level, we design operators to process the entire population at once using the multithread parallel method from Numpy, instead of handling individuals one by one.

## V. EXPERIMENT

Our experiments study the following research questions.

- 1) *RQ1*: How does the performance of our RL-DAS compare with its backbone candidate algorithms, an optimal estimate of traditional AS, and other advanced DE variants (including adaptive, ensemble, and RL-assisted ones)? See Section V-C.
- 2) *RQ2*: How well does RL-DAS generalize to unseen problem instances? See Section V-D.
- 3) *RQ3*: What are the suggested configurations for RL-DAS? See Section V-E.

### A. Benchmark Generation

There are different benchmark test suites proposed in the past two decades to test the performance of EC algorithms, which usually contain 10–20 problem instances. For example, the CEC2021 Competition on Single Objective BBO [14] contains one unimodal problem (F1), three basic problems (F2–F4), three hybrid problems (F5–F7), and three composition problems (F8–F10). For a few existing RL-assisted EC [32], [33], they train their agents on a small number of benchmark problem instances and also test on a small dataset. Though shown improved performance, their learned models may overfit to these fixed instances and lose the generalization on more problem instances. Meanwhile, since the training and test samples are limited, this further hinders their final performance. To this end, this article uses an augmented CEC2021 benchmark that randomly generates a class of instances for each benchmark problem in the CEC2021 (denoted as C1–C10) and requires the agent to achieve generalization performance on the class of instances. The instances in a class are similar problems but use different shifting vectors and rotation matrices. Benefited from the exquisite design of CEC problems, all instances have known optima. Additionally, we adopt a mix-class instance set that contains instances from all problem classes (denoted as C11) to further inspect the generalization ability. Specifically, following the Technical Report [14], we generate rotation matrices with the Gram-Schmidt method [46] and uniformly sample each dimension of the shifting vector in the range  $[-80, 80]$ . The other configurations follow the original settings in CEC2021, such as the selection of subproblems of Hybrid and Composition problems. In this way, we expand each original benchmark into a problem class composed of 2048 instances. The  $K$ -fold cross-validation method is adopted to separate instances into training sets and validation sets, where we use  $K = 4$ , meaning that 512 instances will be taken as validations. Furthermore, another 2048 instances are generated for each problem class to test the performance.

### B. Competitors

In this article, we apply a DRL agent to dynamically select an optimization algorithm at each time step from the three algorithms in the CEC2021 competition: JDE21 [21], NL-SHADE-RSP [23], and MadDE [25]. First, we compare our RL-DAS algorithm with these three algorithms, as well as with a random selection baseline (Rand-DAS), where the RL

agent in RL-DAS is replaced with random selection. Second, to present the difference between DAS and the traditional AS, we compare it with an AS\* algorithm that possesses the performance upper bound of AS. Third, to further validate the effectiveness of RL-DAS, we compare it with some other advanced DEs, including an adaptive DE named EPSDE [18], an ensemble DE named EDEV [26], and three RL-assisted DEs, namely, DEDQN [32], DEDDQN [33], and LDE [37]. All algorithms are implemented in Python. The control parameters and more experiment settings are provided in Section 2 of the supplementary document.

### C. Performance Comparisons

We first compare the optimization results and the optimization speed on 10D and 20D augmented CEC2021 benchmarks. Note that, for all comparisons in this article, we report the cost decent in percentage, and we additionally provide the absolute cost values in Section 4 of our supplementary document.

1) *Comparison With Backbones*: Tables II and III show the performance of algorithms on 10D and 20D problems, respectively, where the upper value is the descent which describes the average descent of costs in percentage (normalized by the best costs in the initial populations, which are consistent among algorithms with random number seeds), and the lower value is FEs describes the average consumed function evaluations to reach the termination criterion. The “+,” “–,” and “ $\approx$ ” indicate the statistical test results that the performance of competing methods is better, worse, and no significant difference with RL-DAS, respectively, according to the Wilcoxon rank-sum test at the 0.05 significance level. The last columns of the two tables summarize the statistical results where the first value counts the cases when RL-DAS significantly outperforms the competitor (the two tests contain at least one “–” while the test on descent is not “+”); the second value counts the cases that there is no significant difference between RL-DAS and the competitor (both tests are “ $\approx$ ”); and the third value counts the cases otherwise. From the results in these tables, we can conclude the following.

- 1) RL-DAS surpasses the Rand-DAS and defeats the backbones on most problems in both 10D and 20D scenarios, which verifies the effectiveness of RL-DAS. Note that the experiment presents the average performance over the instance sets without assuming that RL-DAS defeats the comparison algorithms in all cases.
- 2) For the 10D scenario, while the previously reported results [14] show that the JDE21, NL-SHADE-RSP, and MadDE algorithms can optimize the original benchmark problems toward very low costs, their performance on the augmented benchmarks is far from this. Our results indicate that the algorithms may overfit to the small group of problems provided in the original benchmark, resulting in degraded performance when applied to other shifting and rotation configurations. These complex problems require a better balance between exploration and exploitation to avoid falling into local optima. The three backbone algorithms work with their manual



TABLE II  
COMPARING RL-DAS WITH COMPARISON ALGORITHMS ON SHIFTED AND ROTATED 10D PROBLEMS

Taxonomy		Algorithm	Problem Class										Significance Test (Win/Tie/Loss)
			C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	
DAS	RL-DAS	100.0%	85.97%	95.63%	100.0%	100.0%	99.39%	100.0%	80.48%	90.22%	88.79%	93.94%	- / - / -
		1.127e5	2.000e5	2.000e5	2.000e5	1.996e5	2.000e5	2.000e5	1.821e5	1.882e5	1.914e5	1.903e5	
		100.0%	82.87%	95.40%	100.0%	100.0%	99.33%	100.0%	79.87%	89.91%	88.52%	93.69%	
		1.359e5	2.000e5	2.000e5	1.998e5	1.994e5	2.000e5	2.000e5	1.795e5	1.885e5	1.915e5	1.891e5	
AS	AS*	100.0%	85.63%	95.40%	100.0%	100.0%	99.34%	100.0%	79.86%	89.94%	88.55%	93.71%	7 / 3 / 1
		1.139e5	2.000e5	2.000e5	2.000e5	1.945e5	2.000e5	2.000e5	1.865e5	1.884e5	1.917e5	1.870e5	
Backbones	JDE21	100.0%	73.52%	94.97%	100.0%	99.99%	99.01%	99.99%	78.76%	88.47%	88.43%	92.02%	9 / 2 / 0
		1.914e5	2.000e5	2.000e5	2.000e5	2.000e5	2.000e5	2.000e5	1.935e5	1.954e5	1.954e5	1.975e5	
		100.0%	83.64%	95.39%	100.0%	100.0%	99.34%	100.0%	79.03%	89.77%	87.93%	93.71%	7 / 3 / 1
		1.160e5	2.000e5	2.000e5	2.000e5	1.955e5	2.000e5	2.000e5	1.867e5	1.889e5	1.920e5	1.878e5	
		100.0%	80.74%	95.22%	100.0%	100.0%	97.74%	100.0%	79.41%	89.89%	88.46%	92.86%	9 / 2 / 0
		2.000e5	2.000e5	2.000e5	2.000e5	2.000e5	2.000e5	2.000e5	1.865e5	1.884e5	1.917e5	1.990e5	
Other Advanced DE Variants	Adaptive	EPSDE	100.0%	75.73%	95.48%	100.0%	99.23%	100.0%	76.44%	87.21%	90.18%	92.43%	7 / 3 / 1
		1.476e5	2.000e5	2.000e5	2.000e5	1.967e5	2.000e5	2.000e5	1.834e5	1.896e5	1.983e5	1.968e5	
	Ensemble	EDEV	100.0%	77.45%	95.53%	100.0%	99.27%	100.0%	77.37%	87.65%	91.06%	93.02%	7 / 3 / 1
		1.402e5	2.000e5	2.000e5	2.000e5	1.960e5	2.000e5	2.000e5	1.827e5	1.889e5	1.980e5	1.909e5	
	RL Assisted	DEDQN	100.0%	71.65%	92.43%	100.0%	99.99%	100.0%	76.09%	83.42%	87.85%	91.36%	9 / 2 / 0
		1.964e5	2.000e5	2.000e5	2.000e5	2.000e5	2.000e5	2.000e5	1.875e5	1.904e5	1.967e5	1.970e5	
		DEDDQN	100.0%	71.98%	93.23%	100.0%	99.99%	100.0%	76.85%	84.52%	88.07%	91.26%	9 / 2 / 0
		1.989e5	2.000e5	2.000e5	2.000e5	2.000e5	2.000e5	2.000e5	1.864e5	1.890e5	1.956e5	1.969e5	
	LDE	100.0%	76.44%	95.39%	100.0%	100.0%	98.91%	100.0%	77.23%	83.74%	88.28%	92.20%	9 / 2 / 0
		1.564e5	2.000e5	2.000e5	2.000e5	2.000e5	2.000e5	2.000e5	1.861e5	1.895e5	1.933e5	1.960e5	

TABLE III  
COMPARING RL-DAS WITH COMPARISON ALGORITHMS ON SHIFTED AND ROTATED 20D PROBLEMS

Taxonomy		Algorithm	Problem Class										Significance Test (Win/Tie/Loss)	
			C1	C2	C3	C4	C5	C6	C7	C8	C9	C10		C11
DAS		RL-DAS	100.0%	76.54%	96.89%	100.0%	100.0%	97.91%	100.0%	79.59%	87.55%	89.74%	92.87%	- / - / -
			3.547e5	1.000e6	1.000e6	1.000e6	1.000e6	1.000e6	1.000e6	8.489e5	9.211e5	9.112e5	9.231e5	
			100.0% $\approx$ 6.856e5	73.70% $\approx$ 1.000e6	96.83% $\approx$ 1.000e6	100.0% $\approx$ 1.000e6	100.0% $\approx$ 1.000e6	96.51% $\approx$ 1.000e6	100.0% $\approx$ 1.000e6	79.14% $\approx$ 9.653e5	87.03% $\approx$ 9.456e5	89.48% $\approx$ 9.520e5	92.35% $\approx$ 9.347e5	
AS		AS*	100.0%	75.85%	96.49%	100.0%	100.0%	97.62%	100.0%	78.86%	87.13%	89.39%	92.57%	8 / 3 / 0
			3.635e5	1.000e6	1.000e6	1.000e6	1.000e6	1.000e6	1.000e6	8.763e5	9.137e5	9.276e5	8.973e5	
Backbones		JDE21	100.0%	62.73%	96.24%	100.0%	100.0%	95.05%	100.0%	77.72%	86.55%	89.32%	90.86%	8 / 3 / 0
			9.573e5	1.000e6	1.000e6	1.000e6	1.000e6	1.000e6	1.000e6	9.619e5	9.432e5	9.241e5	9.880e5	
		MadDE	100.0%	75.76%	96.53%	100.0%	100.0%	97.62%	100.0%	78.21%	86.66%	89.03%	92.64%	8 / 3 / 0
			3.681e5	1.000e6	1.000e6	1.000e6	1.000e6	1.000e6	1.000e6	8.787e5	9.202e5	9.486e5	9.022e5	
		NL-SHADE-RSP	100.0%	70.57%	95.57%	100.0%	100.0%	92.94%	100.0%	77.13%	86.35%	87.36%	91.11%	8 / 3 / 0
			1.000e6	1.000e6	1.000e6	1.000e6	1.000e6	1.000e6	1.000e6	9.653e5	9.505e5	9.641e5	9.923e5	
Other Advanced DE Variants	Adaptive	EPSDE	100.0%	64.31%	96.38%	100.0%	100.0%	97.24%	100.0%	78.83%	86.45%	89.95%	91.52%	7 / 3 / 1
			4.724e5	1.000e6	1.000e6	1.000e6	1.000e6	1.000e6	1.000e6	9.588e5	9.462e5	9.031e5	9.471e5	
	Ensemble	EDEV	100.0%	66.16%	96.43%	100.0%	100.0%	97.32%	100.0%	78.73%	86.56%	89.97%	91.34%	7 / 3 / 1
			4.545e5	1.000e6	1.000e6	1.000e6	1.000e6	1.000e6	1.000e6	9.499e5	9.351e5	8.989e5	9.414e5	
	RL Assisted	DEDQN	100.0%	63.74%	95.38%	100.0%	100.0%	96.11%	100.0%	75.15%	85.53%	88.01%	90.90%	8 / 3 / 0
			9.945e5	1.000e6	1.000e6	1.000e6	1.000e6	1.000e6	1.000e6	9.812e5	9.687e5	9.544e5	9.962e5	
		DEDDQN	100.0%	62.19%	95.15%	100.0%	100.0%	96.54%	100.0%	75.91%	86.03%	88.01%	91.05%	8 / 3 / 0
			9.813e5	1.000e6	1.000e6	1.000e6	1.000e6	1.000e6	1.000e6	9.735e5	9.625e5	9.519e5	9.950e5	
		LDE	100.0%	61.22%	95.96%	100.0%	100.0%	97.54%	100.0%	77.15%	86.30%	88.52%	91.33%	8 / 3 / 0
			5.642e5	1.000e6	1.000e6	1.000e6	1.000e6	1.000e6	1.000e6	9.287e5	9.530e5	9.433e5	9.487e5	

exploration–exploitation balance methods, which give them unique searching characteristics and make them winners of the CEC2021 competition. However, these manual designs are based on a small group of benchmark problems and fail to show enough adaptiveness in more test instances. On the contrary, RL-DAS employs a DRL agent to dynamically switch these algorithms to achieve comprehensively better performance.

- For the 20D problems shown in Table III, RL-DAS still outperforms the others. The increased problem space makes the optimization more difficult, and our relative performance over the competitors is even better than that on 10D problems. The results indicate that dimensions do not affect the RL-DAS much, the conclusions on 10D problems are still applicable to the 20D benchmark.
- Furthermore, an intriguing observation is the improved performance of the Rand-DAS over the three backbone algorithms. This further validates the importance of DAS over traditional AS setting: a single handcrafted algorithm can hardly be efficient in solving all problem cases (as indicated by the NFL theorem), and even a random combination of a few algorithms has the potential to achieve complementary performance. Nevertheless, the comparison between RL-DAS and Rand-DAS shows that our RL agent further exploits this potential and shows significant improvement over the random agent.

2) *Comparison Between DAS and AS*: The DAS and backbones parts of Tables II and III verify that RL-DAS outperforms the compared single algorithm. Then, a question comes out: can RL-DAS defeat other methods that combine these single algorithms, for example, by the AS? To figure out the answer, we introduce a competitor that takes the best result among the three backbone algorithms as its performance, namely, AS\*, to simulate the performance upper bound of AS methods. For each problem instance  $i$ , the result of AS\* is calculated by

$$AS^*(i) = \max(JDE21(i), MadDE(i), NL-SHADE-RSP(i)) \quad (20)$$

where JDE21( $i$ ) indicates the optimization result of JDE21 on instance  $i$ , and so do the items for MadDE and NL-SHADE-RSP. The performance comparison is shown in the AS parts of Tables II and III.

It can be seen that RL-DAS outperforms AS\* in most of the problems. AS methods employ a single candidate algorithm for each problem instance, therefore the upper bound of its performance can not exceed the best performance among all candidate algorithms, let alone that practical AS methods usually require external evaluation resources to select an algorithm [6] which will inevitably impact the performance. Differently, the DAS methods select an algorithm at each interval which has an opportunity to present a complementary

TABLE IV  
COMPUTATIONAL COST COMPARISON

Algorithm	JDE21	MadDE	NL-SHADE-RSP	EPSDE	EDEV
Time (s)	0.748	0.905	0.791	0.534	0.717
Algorithm	RL-DAS	DEDQN	DEDDQN	LDE	
Time (s)	1.434	0.986	66.891	1.263	

performance and breaks through the single best candidate algorithm. The outstanding performance of Rand-DAS in Tables II and III prove its potential. The employment of a DRL agent ensures wise selection thus RL-DAS defeats AS\* significantly.

3) *Comparison With Other Advanced DE Variants:* The last part in Tables II and III present the comparison between our RL-DAS and other advanced DE. For the sake of fairness, the three RL-assisted methods are trained on the same training set as RL-DAS and update their models by equal steps.

RL-DAS demonstrates a significant advantage over adaptive and ensemble algorithms for both 10D and 20D problems. Additionally, for the three methods utilizing RL agents, namely, DEDQN, DEDDQN, and LDE, RL-DAS also outperforms them. This can be attributed to RL-DAS's integration of three advanced algorithms, which incorporates the existing expert knowledge for a promising performance.

4) *Runtime Comparison:* Table IV presents the average computational time of different algorithms on 10D C2 (Schwefel) testing set problems. The upper row represents traditional non-learning algorithms, while the lower row presents the RL-assisted algorithms including our RL-DAS. It is evident that the computational costs of most DE algorithms are comparable, with MadDE being relatively slower due to its use of more operators and parameter adaptive mechanisms. RL-assisted algorithms, in general, require more time for network inference. Among them, DEDDQN takes the longest time, attributed to its costly and complex 99D state representation and one-by-one operator selection for each individual. RL-DAS, on the other hand, completes optimization in 1.434 s for 200 000 FEs. While it requires more time for network inference and algorithm switching compared to JDE21, MadDE, and NL-SHADE-RSP, the additional computational cost is generally acceptable regarding the state-of-the-art optimization accuracy of RL-DAS.

In summary, RL-DAS achieves state-of-the-art optimization performance, it outperforms not only the backbones but also the advanced DE variants, including adaptive DE and RL-assisted DE, while incurring an additional acceptable computational cost. To unveil the secret behind this performance, we conduct an in-depth analysis of the learned policy, as detailed in Section 3 of our supplementary document.

#### D. Zero-Shot Knowledge Transfer

In this section, we conduct two types of zero-shot knowledge transfer experiments. First, we choose models from the three problem classes which have significant performance differences as reported in Section V-C: the unimodal problem C1 (Bent Cigar), the basic multimodal problem C2 (Schwefel) and the mix-class problems C11, and then zero-shot transfer

TABLE V  
TRANSFER RESULTS FROM ONE CLASS OF PROBLEMS TO OTHERS

Problem Class	RL-DAS Zero-shot (C1)	RL-DAS Zero-shot (C2)	RL-DAS Zero-shot (C11)	RL-DAS
C1		100.0% $\approx$ 1.046e5 +	100.0% $\approx$ 1.215e5 -	100.0% 1.127e5
C2	84.32% - 2.000e5 $\approx$		83.69% - 2.000e5 $\approx$	85.97% 2.000e5
C3	95.67% $\approx$ 2.000e5 $\approx$	95.61% $\approx$ 2.000e5 $\approx$	95.86% + 2.000e5 $\approx$	95.63% 2.000e5
C4	100.0% $\approx$ 1.999e5 $\approx$	100.0% $\approx$ 1.996e5 +	100.0% $\approx$ 2.000e5 $\approx$	100.0% 2.000e5
C5	100.0% $\approx$ 1.996e5 $\approx$	100.0% $\approx$ 1.986e5 +	100.0% $\approx$ 1.997e5 $\approx$	100.0% 1.996e5
C6	99.37% $\approx$ 2.000e5 $\approx$	99.56% + 2.000e5 $\approx$	99.38% $\approx$ 2.000e5 $\approx$	99.39% 2.000e5
C7	100.0% $\approx$ 2.000e5 $\approx$	100.0% $\approx$ 2.000e5 $\approx$	100.0% $\approx$ 2.000e5 $\approx$	100.0% 2.000e5
C8	80.63% + 1.801e5 +	80.27% - 1.811e5 +	80.29% - 1.830e5 -	80.48% 1.821e5
C9	90.14% - 1.868e5 +	89.73% - 1.873e5 +	89.93% - 1.889e5 -	90.18% 1.882e5
C10	88.32% - 1.909e5 +	88.81% $\approx$ 1.909e5 +	88.59% - 1.921e5 -	88.73% 1.914e5
	2 / 5 / 2	2 / 2 / 5	3 / 4 / 1	- / - / -

TABLE VI  
TRANSFER RESULTS UNDER DIFFERENT PROBLEM CLASS PARTITIONS

Transfer Setting	NL-SHADE-RSP	MadDE	JDE21	RL-DAS (Zero-shot)
TS1	88.38% - 2.000e5 -	92.62% - 1.999e5 -	89.87% - 2.000e5 -	92.86% 1.991e5
TS2	84.88% - 2.000e5 -	89.47% - 1.961e5 +	86.12% - 1.985e5 -	89.62% 1.969e5
TS3	90.76% - 2.000e5 -	91.47% - 1.951e5 -	90.86% - 1.988e5 -	91.65% 1.917e5
	3 / 0 / 0	3 / 0 / 0	3 / 0 / 0	- / - / -

them to the other CEC problems. The results are shown in Table V. It can be seen that RL-DAS successfully transfers its learned knowledge to unseen problems, which allows the models from C1, C2, and C11 achieve close results to the original RL-DAS.

In the second experiment, we establish different transfer settings, denoted as TS1–TS3, by partitioning the ten classes of problems into training and testing sets with varying proportions: 2:8 (F1–F2 for training, F3–F10 for testing), 5:5 (F1–F5 for training, F6–F10 for testing), and 8:2 (F1–F8 for training, F9–F10 for testing), respectively. The training and testing sets, with instances mixed from different problem classes, are generated in the same way as C11. The results are shown in Table VI, where the three backbone algorithms serve as references. RL-DAS outperforms the baselines on most of transfer settings in Table VI. It demonstrates promising generalization ability, transferring the knowledge learned from a small training problem set to a much larger and more complex problem set.

The results in the two tables collectively show that the RL agent captures the intrinsic characteristics of problems and the searching strategies of the backbone algorithms. This enables the agent to generalize its knowledge to unseen problems and achieve state-of-the-art performance.

#### E. Ablation Study and Hyperparameter Research

1) *Ablation Study:* In order to verify the necessity of the framework components, we conducted ablation studies on the state features and the context memory. For the features, we removed the embedding and concatenation of the LA feature (named RL-DAS w/o LA) and the AH feature (named RL-DAS w/o AH), respectively, and tested them on 10D problems. The ablation study that removes both features is equivalent to the “Rand-DAS” condition studied in previous experiments, as the model cannot function without input. Besides, we ablate the context memory (named RL-DAS w/o Context) to investigate the effect of the warm start procedure. In this scenario, the parameter memories, elite archives and other adaption mechanisms are cleaned and reinitialized at the beginning of each internal.

TABLE VII  
ABLATION STUDY PERFORMANCE

Problem Class	RL-DAS w/o LA	RL-DAS w/o AH	RL-DAS w/o Context	RL-DAS
C1	100.0% $\approx$ 1.425e5	100.0% $\approx$ 1.383e5	99.96% $\approx$ 2.000e5	100.0% 1.127e5
C2	82.29% $\approx$ 2.000e5	82.73% $\approx$ 2.000e5	56.34% $\approx$ 2.000e5	85.97% 2.000e5
C3	95.31% $\approx$ 2.000e5	95.36% $\approx$ 2.000e5	90.17% $\approx$ 2.000e5	95.63% 2.000e5
C4	100.0% $\approx$ 2.000e5	100.0% $\approx$ 2.000e5	99.96% $\approx$ 2.000e5	100.0% 2.000e5
C5	100.0% $\approx$ 2.000e5	100.0% $\approx$ 2.000e5	99.97% $\approx$ 2.000e5	100.0% 1.996e5
C6	98.96% $\approx$ 2.000e5	98.91% $\approx$ 2.000e5	95.08% $\approx$ 2.000e5	99.39% 2.000e5
C7	99.99% $\approx$ 2.000e5	100.0% $\approx$ 2.000e5	99.21% $\approx$ 2.000e5	100.0% 2.000e5
C8	79.32% $\approx$ 1.851e5	79.77% $\approx$ 1.873e5	68.43% $\approx$ 2.000e5	80.48% 1.821e5
C9	89.52% $\approx$ 1.905e5	89.86% $\approx$ 1.912e5	80.28% $\approx$ 2.000e5	90.22% 1.882e5
C10	88.24% $\approx$ 1.916e5	87.76% $\approx$ 1.933e5	77.67% $\approx$ 2.000e5	88.79% 1.914e5
C11	92.95% $\approx$ 1.939e5	93.16% $\approx$ 1.948e5	84.27% $\approx$ 2.000e5	93.94% 1.903e5
	10 / 1 / 0	9 / 2 / 0	11 / 0 / 0	- / - / -

Table VII shows the results. As can be observed, the significant role of the two features is highlighted by the poor performance if they are removed. For the ablation of the context memory, the results also demonstrate a significant degradation in performance. Without historical memory restoration, algorithms are compelled to optimize the population from scratch in each period, hindering comprehensive algorithm cooperation

2) *Reward Design*: RL-DAS adopts a reward mechanism that combines absolute cost descent and an adaptation term based on the convergence speed. Within this part, we thoroughly examine five different reward designs, by taking the performance of RL-DAS on 10D problems as an example.

1) *Absolute Cost Descent Reward*: The decline of the optimal cost of the population at time step  $t$  normalized by the optimal cost of the initial population

$$r1 = \frac{cost_{t-1}^* - cost_t^*}{cost_0^*}. \quad (21)$$

2) *Relative Cost Descent Ratio*: The optimal cost decline normalized by the optimal cost in the previous generation

$$r2 = \frac{cost_{t-1}^* - cost_t^*}{cost_{t-1}^*}. \quad (22)$$

3) *Zero-One Reward*: The reward will be 1 if the optimal cost of the population decreased. Otherwise, it is 0

$$r3 = \begin{cases} 1, & \text{if } cost_{t-1}^* - cost_t^* > 0 \\ 0, & \text{otherwise.} \end{cases} \quad (23)$$

4) *Restricted Zero-One Reward*: A restriction is introduced to the zero-one reward that the reward will be 1 only if the difference between costs is larger than a threshold (2.5%)

$$r4 = \begin{cases} 1, & \text{if } \frac{cost_{t-1}^* - cost_t^*}{cost_{t-1}^*} > 0.025 \\ 0, & \text{otherwise.} \end{cases} \quad (24)$$

5) *Adjusted Cost Descent Reward*: Apply an external term on  $r1$  to include the consumed FEs to reach the termination. It is shown in Eq. (13) and applied in RL-DAS.

The comparison results for 10D problems are presented in Table VIII. Our reward explicitly considers both cost and speed, offering valuable guidance to the agent. It is evident from the table that our reward outperforms others on the majority of problems, particularly in terms of optimization speed. In contrast, alternative reward functions, such as  $r1$ , offer no differentiation on FE consumption, resulting in identical

TABLE VIII  
PERFORMANCE UNDER DIFFERENT REWARD SCHEMES

Problem Class	Reward Type				
	r1	r2	r3	r4	Ours
C1	100.0% $\approx$ 1.343e5	100.0% $\approx$ 1.035e5	100.0% $\approx$ 1.360e5	100.0% $\approx$ 1.359e5	100.0% $\approx$ 1.127e5
C2	85.79% $\approx$ 2.000e5	85.73% $\approx$ 2.000e5	85.46% $\approx$ 2.000e5	83.96% $\approx$ 2.000e5	85.97% $\approx$ 2.000e5
C3	95.36% $\approx$ 2.000e5	95.37% $\approx$ 2.000e5	95.33% $\approx$ 2.000e5	94.14% $\approx$ 2.000e5	95.63% $\approx$ 2.000e5
C4	100.0% $\approx$ 2.000e5	100.0% $\approx$ 2.000e5	100.0% $\approx$ 2.000e5	100.0% $\approx$ 2.000e5	100.0% $\approx$ 2.000e5
C5	100.0% $\approx$ 2.000e5	100.0% $\approx$ 1.999e5	100.0% $\approx$ 2.000e5	100.0% $\approx$ 2.000e5	100.0% $\approx$ 1.996e5
C6	99.01% $\approx$ 2.000e5	98.46% $\approx$ 2.000e5	98.68% $\approx$ 2.000e5	97.91% $\approx$ 2.000e5	99.39% $\approx$ 2.000e5
C7	100.0% $\approx$ 2.000e5	100.0% $\approx$ 2.000e5	100.0% $\approx$ 2.000e5	100.0% $\approx$ 2.000e5	100.0% $\approx$ 2.000e5
C8	80.36% $\approx$ 1.923e5	80.01% $\approx$ 1.798e5	79.77% $\approx$ 1.891e5	80.43% $\approx$ 1.957e5	80.48% $\approx$ 1.821e5
C9	90.20% $\approx$ 1.924e5	89.86% $\approx$ 1.880e5	89.73% $\approx$ 1.916e5	89.17% $\approx$ 1.943e5	90.22% $\approx$ 1.882e5
C10	88.61% $\approx$ 1.951e5	88.55% $\approx$ 1.916e5	88.16% $\approx$ 1.943e5	87.63% $\approx$ 1.949e5	88.79% $\approx$ 1.914e5
C11	93.65% $\approx$ 1.924e5	93.56% $\approx$ 1.886e5	93.52% $\approx$ 1.921e5	93.07% $\approx$ 1.941e5	93.94% $\approx$ 1.903e5
	9 / 2 / 0	8 / 2 / 1	9 / 2 / 0	9 / 2 / 0	- / - / -

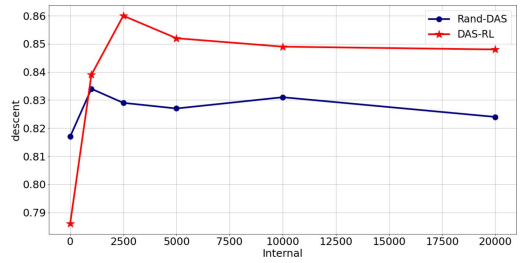


Fig. 4. Performance change along schedule intervals.

rewards for fast or slow convergence. Consequently, the agent lacks motivation to expedite optimization, leading to relatively poor performance.

3) *Schedule Interval*: In the previous experiment, we fixed the interval on 10D problems to 2500 FEs. However, different intervals can have an impact on performance. In this section, we present the performance of RL-DAS and the Rand-DAS for intervals with values of 0, 1000, 2500, 5000, 10000, and 20000 on 10D Schwefel. The "0" interval indicates switching at every EC generation. The performance is shown in Fig. 4.

The performance of RL-DAS and Rand-DAS shows similar tendency. Among these intervals, RL-DAS with 2500 interval achieves the best performance. The smallest interval, 0, presents a significantly poor performance due to the algorithm's integrity being compromised by too frequent switching and a large number of feature samplings (which consume a lot of additional evaluations). These factors make it significantly worse than Rand-DAS which has no feature samplings. The performance of intervals larger than 2500 shows a gentle decrease as they lose flexibility but retain algorithm integrity and use less feature sampling. Therefore, an interval of 2500 is suggested in this article for RL-DAS.

## VI. CONCLUSION

This article proposed the RL-DAS framework to select the optimal algorithm at different optimization stages for given problems. We characterized the optimization state space through analyzing the fitness landscape and algorithm history. We then designed a deep neural network to infer the best



action (namely, algorithm selection) for each state, which was trained in the actor–critic-based PPO manner. Meanwhile, the algorithm context memory mechanism was deployed to support the warm start of different algorithms. RL-DAS allows for seamless switching among candidate algorithms to optimize the problem in a dynamic-online fashion, while it is simple and generic and has the potential to boost many EC algorithms.

We applied the framework to a group of DE algorithms for investigation. Experimental results showed that RL-DAS not only improved the overall optimization performance but also exhibited desirable generalization ability across different problem classes. The detailed analysis also verified the importance of DAS, which provided new opportunities to boost the traditional AS method. There are still some limitations such as the manual state representation and challenging integration of complex algorithms. Therefore, considerable future efforts are warranted to explore this area, such as automatic state representation and the inclusion of more candidate algorithms.

## REFERENCES

- [1] A. Slowik and H. Kwasnicka, "Evolutionary algorithms and their applications to engineering problems," *Neural Comput. Appl.*, vol. 32, no. 16, pp. 12363–12379, 2020.
- [2] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis With Applications to Biology, Control, and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992.
- [3] R. Storn and K. Price, "Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optim.*, vol. 11, no. 4, p. 341, 1997.
- [4] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. Int. Conf. Neural Netw.*, vol. 4, 1995, pp. 1942–1948.
- [5] D. H. Wolpert and W. G. Macready, "No free lunch theorems for search," Santa Fe Inst., Santa Fe, NM, USA, document SFI-TR-95-02-010, 1995.
- [6] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann, "Automated algorithm selection: Survey and perspectives," *Evol. Comput.*, vol. 27, no. 1, pp. 3–45, 2019.
- [7] B. A. Huberman, R. M. Lukose, and T. Hogg, "An economics approach to hard computational problems," *Science*, vol. 275, no. 5296, pp. 51–54, 1997.
- [8] C. P. Gomes and B. Selman, "Algorithm portfolios," *Artif. Intell.*, vol. 126, nos. 1–2, pp. 43–62, 2001.
- [9] C. Ansótegui, M. Sellmann, and K. Tierney, "A gender-based genetic algorithm for the automatic configuration of algorithms," in *Proc. Int. Conf. Princ. Pract. Constr. Program.*, 2009, pp. 142–157.
- [10] K. Xue et al., "Multi-agent dynamic algorithm configuration," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, pp. 1–28.
- [11] A. Biedenkapp, H. F. Bozkurt, T. Eimer, F. Hutter, and M. Lindauer, "Dynamic algorithm configuration: Foundation of a new meta-algorithmic framework," in *Proc. 24th Eur. Conf. Artif. Intell.*, 2020, pp. 427–434.
- [12] B. Bischl, O. Mersmann, H. Trautmann, and M. Preuß, "Algorithm selection based on exploratory landscape analysis and cost-sensitive learning," in *Proc. 14th Annu. Conf. Genetic Evol. Comput.*, 2012, pp. 313–320.
- [13] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, [arXiv:1707.06347](https://arxiv.org/abs/1707.06347).
- [14] A. W. Mohamed, A. A. Hadi, A. K. Mohamed, P. Agrawal, A. Kumar, and P. N. Suganthan, "Problem definitions and evaluation criteria for the CEC 2021 on single objective bound constrained numerical optimization," Nanyang Technol. Univ., Singapore, Nov. 2020.
- [15] W. Gong, Y. Wang, S. Cai, and L. Wang, "Finding multiple roots of nonlinear equation systems via a repulsion-based adaptive differential evolution," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 50, no. 4, pp. 1499–1513, Apr. 2020.
- [16] Y.-J. Gong, Y.-W. Liu, Y. Lin, W.-N. Chen, and J. Zhang, "Real-time taxi-passenger matching using a differential evolutionary fuzzy controller," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 51, no. 5, pp. 2712–2725, May 2021.
- [17] W. Deng, J. Xu, X.-Z. Gao, and H. Zhao, "An enhanced MSIQDE algorithm with novel multiple strategies for global optimization problems," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 52, no. 3, pp. 1578–1587, Mar. 2022.
- [18] R. Mallipeddi, P. N. Suganthan, Q.-K. Pan, and M. F. Tasgetiren, "Differential evolution algorithm with ensemble of parameters and mutation strategies," *Appl. Soft Comput.*, vol. 11, no. 2, pp. 1679–1696, 2011.
- [19] Y. Li, S. Wang, H. Yang, H. Chen, and B. Yang, "Enhancing differential evolution algorithm using leader-adjoint populations," *Inf. Sci.*, vol. 622, pp. 235–268, Apr. 2023.
- [20] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer, "Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems," *IEEE Trans. Evol. Comput.*, vol. 10, no. 6, pp. 646–657, Dec. 2006.
- [21] J. Brest, M. S. Maučec, and B. Bošković, "Self-adaptive differential evolution algorithm with population size reduction for single objective bound-constrained optimization: Algorithm j21," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2021, pp. 817–824.
- [22] R. Tanabe and A. Fukunaga, "Success-history based parameter adaptation for differential evolution," in *Proc. IEEE Congr. Evol. Comput.*, 2013, pp. 71–78.
- [23] V. Stanovov, S. Akhmedova, and E. Semenkin, "NL-SHADE-RSP algorithm with adaptive archive and selective pressure for CEC 2021 numerical optimization," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2021, pp. 809–816.
- [24] V. Stanovov, S. Akhmedova, and E. Semenkin, "NL-SHADE-LBC algorithm with linear parameter adaptation bias change for CEC 2022 numerical optimization," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2022, pp. 1–8.
- [25] S. Biswas, D. Saha, S. De, A. D. Cobb, S. Das, and B. A. Jalaian, "Improving differential evolution through Bayesian hyperparameter optimization," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2021, pp. 832–840.
- [26] G. Wu, X. Shen, H. Li, H. Chen, A. Lin, and P. N. Suganthan, "Ensemble of differential evolution variants," *Inf. Sci.*, vol. 423, pp. 172–186, Jan. 2018.
- [27] J. Zhang and A. C. Sanderson, "JADE: Adaptive differential evolution with optional external archive," *IEEE Trans. Evol. Comput.*, vol. 13, no. 5, pp. 945–958, Oct. 2009.
- [28] G. Li et al., "A novel hybrid differential evolution algorithm with modified CoDE and JADE," *Appl. Soft Comput.*, vol. 47, pp. 577–599, Oct. 2016.
- [29] S. Li, W. Gong, L. Wang, and Q. Gu, "Evolutionary multitasking via reinforcement learning," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 8, no. 1, pp. 762–775, Feb. 2024.
- [30] Z. Liao, W. Gong, and S. Li, "Two-stage reinforcement learning-based differential evolution for solving nonlinear equations," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 53, no. 7, pp. 4279–4290, Jul. 2023.
- [31] Q. Yang, S.-C. Chu, J.-S. Pan, J.-H. Chou, and J. Watada, "Dynamic multi-strategy integrated differential evolution algorithm based on reinforcement learning for optimization problems," *Complex Intell. Syst.*, pp. 1–33, Oct. 2023.
- [32] Z. Tan and K. Li, "Differential evolution with mixed mutation strategy based on deep reinforcement learning," *Appl. Soft Comput.*, vol. 111, Nov. 2021, Art. no. 107678.
- [33] M. Sharma, A. Komninos, M. López-Ibáñez, and D. Kazakov, "Deep reinforcement learning based parameter control in differential evolution," in *Proc. Genetic Evol. Comput. Conf.*, 2019, pp. 709–717.
- [34] C. J. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, pp. 279–292, May 1992.
- [35] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [36] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI Conf. Artif. Intell.*, vol. 30, 2016, pp. 2094–2100.
- [37] J. Sun, X. Liu, T. Bäck, and Z. Xu, "Learning adaptive differential evolution algorithm from optimization experiences by policy gradient," *IEEE Trans. Evol. Comput.*, vol. 25, no. 4, pp. 666–680, Aug. 2021.
- [38] A. Graves, "Long short-term memory," *Supervised Sequence Labelling Recurrent Neural Network*. Berlin, Germany: Springer, pp. 37–45, 2012.
- [39] Z. Tan, Y. Tang, K. Li, H. Huang, and S. Luo, "Differential evolution with hybrid parameters and mutation strategies based on reinforcement learning," *Swarm Evol. Comput.*, vol. 75, Dec. 2022, Art. no. 101194.
- [40] M. Tomassini, L. Vanneschi, P. Collard, and M. Clergue, "A study of fitness distance correlation as a difficulty measure in genetic programming," *Evol. Comput.*, vol. 13, no. 2, pp. 213–239, 2005.

- [41] M. Lunacek and D. Whitley, "The dispersion metric and the CMA evolution strategy," in *Proc. 8th Annu. Conf. Genetic Evol. Comput.*, 2006, pp. 477–484.
- [42] L. Vanneschi, M. Clergue, P. Collard, M. Tomassini, and S. Vérel, "Fitness clouds and problem hardness in genetic programming," in *Proc. Genetic Evol. Comput. Conf.*, 2004, pp. 690–701.
- [43] L. Vanneschi, M. Tomassini, P. Collard, S. Vérel, Y. Pirola, and G. Mauri, "A comprehensive view of fitness landscapes with neutrality and fitness clouds," in *Proc. Eur. Conf. Genetic Program.*, 2007, pp. 241–250.
- [44] M. Wang, B. Li, G. Zhang, and X. Yao, "Population evolvability: Dynamic fitness landscape analysis for population-based metaheuristic algorithms," *IEEE Trans. Evol. Comput.*, vol. 22, no. 4, pp. 550–563, Aug. 2018.
- [45] G. Brockman et al., "OpenAI gym," 2016, *arXiv:1606.01540*.
- [46] D. Hilbert and E. Schmidt, "Zur Theorie der linearen und nichtlinearen Integralgleichungen: I. Teil: Entwicklung willkürlicher Funktionen nach Systemen vorgeschriebener," in *Integralgleichungen und Gleichungen mit unendlich vielen Unbekannten*. Wiesbaden, Germany: Springer Vieweg Verlag, pp. 190–233, 1989.



**Hongshu Guo** received the B.Eng. degree in computer science from the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China, in 2022, where he is currently pursuing the Ph.D. degree.

His research interests include deep reinforcement learning and evolutionary computing.



**Yining Ma** received the B.E. degree in computer science from the South China University of Technology, Guangzhou, China, in 2019. He is currently pursuing the Ph.D. degree in industrial systems engineering with the National University of Singapore, Singapore.

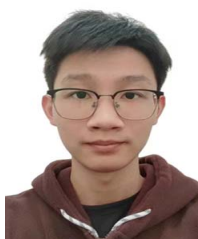
He is currently a Researcher with the College of Computing and Data Science, Nanyang Technological University, Singapore. His research focuses on the intersection of artificial intelligence (AI) and optimization, also known as the field of

learning to optimize, and extends to broader AI domains, such as federated learning, multiagent systems, and computational intelligence.



**Zeyuan Ma** received the B.Eng. degree in computer science from the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China, in 2022, where he is currently pursuing the Ph.D. degree.

His research interests include deep reinforcement learning, continuous black-box optimization, and evolutionary computing.



**Jiacheng Chen** is currently pursuing the B.Eng. degree in computer science with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China.

His research interests include deep reinforcement learning and evolutionary computing.



**Xinglin Zhang** (Member, IEEE) received the B.E. degree in computer science and engineering from the School of Software, Sun Yat-sen University, Guangzhou, China, in 2010, and the Ph.D. degree in computer science and engineering from the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, in 2014.

He is currently an Associate Professor with the South China University of Technology, Guangzhou. His research interests include mobile crowdsensing,

edge computing, and federated learning.

Dr. Zhang is a member of ACM.



**Zhiguang Cao** (Member, IEEE) received the B.Eng. degree in automation from the Guangdong University of Technology, Guangzhou, China, in 2009, the M.Sc. degree in signal processing from Nanyang Technological University (NTU), Singapore, in 2012, and the Ph.D. degree in computer science from the Interdisciplinary Graduate School, NTU, in 2017.

He was a Research Fellow with the Energy Research Institute @ NTU (ERI@N), Singapore, a Research Assistant Professor with the Department

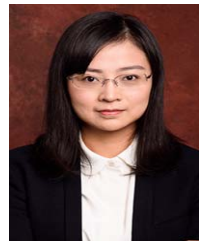
of Industrial Systems Engineering and Management, National University of Singapore, Singapore, and a Scientist with the Agency for Science Technology and Research, Singapore. He joins the School of Computing and Information Systems, Singapore Management University, Singapore, as an Assistant Professor. His research interests focus on learning to optimize.



**Jun Zhang** (Fellow, IEEE) received the Ph.D. degree in electrical engineering from the City University of Hong Kong, Hong Kong, in 2002.

His research contributions span over 500 peer-reviewed publications, of which more than 220 appear in IEEE TRANSACTIONS. His research interests include computational intelligence and evolutionary computation.

Prof. Zhang currently serves as an Associate Editor for the IEEE TRANSACTIONS ON ARTIFICIAL INTELLIGENCE and the IEEE TRANSACTIONS ON CYBERNETICS.



**Yue-Jiao Gong** (Senior Member, IEEE) received the B.S. and Ph.D. degrees in computer science from Sun Yat-sen University, Guangzhou, China, in 2010 and 2014, respectively.

She currently holds the position of Full Professor with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China. Her research interests encompass optimization methods based on swarm intelligence, deep learning, reinforcement learning, and their applications in smart cities and

intelligent transportation. She has published over 100 papers, including more than 50 ACM/IEEE TRANSACTIONS papers, in her research areas.

Dr. Gong was a recipient of the Pearl River Young Scholar from the Guangdong Education Department in 2017, and the Guangdong Natural Science Funds for Distinguished Young Scholars in 2022.