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

Hongshu Guo ⬥, Yining Ma ⬥, Zeyuan Ma ⬥, Jiacheng Chen ⬥, Xinglin Zhang ⬥, *Member, IEEE*,
Zhiguang Cao ⬥, Jun Zhang ⬥, *Fellow, IEEE* and Yue-Jiao Gong ⬥, *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 paper 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.

Keywords– Algorithm selection, deep reinforcement learning, meta-black-box optimization, black-box optimization, differential evolution

1 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 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: Algorithm Portfolio (AP), Algorithm Configuration (AC) and Algorithm Selection (AS).

AP, also known as *parallel algorithm portfolio*, 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 et al. [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, Ansotegui et al. [9] took advantage of the iterated searching algorithms to search proper parameter 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].

Figure 1: The overall structure of RL-DAS.

Table 1: The 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

One way to relieve the above issue of AC is to reduce the configuration space by using expert knowledge to pre-determine 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 algorithm selection. 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 paper proposes a deep **R**einforcement **L**earning based **D**ynamic **A**lgorithm **S**election (**RL-DAS**) framework, presenting following key contributions.

1. We extend the AS to a Dynamic Algorithm Selection (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 Reinforcement Learning (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.

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 rest of this paper is organized as follows: Section 2 presents the preliminaries on DRL and DE. Section 3 introduces the detail of RL-DAS. Section 4 is the proof-of-principle study of applying RL-DAS on advanced DE algorithms. The experimental results are depicted in Section 5 with detailed analysis, followed by a conclusion in Section 6.

2 Preliminaries and Related Works

In this section, we briefly introduce the Markov Decision Process and the advanced variants of DE. Table 1 summarizes the abbreviation indices for easy reference.

2.1 Markov Decision Process and Policy Gradient

A Markov Decision Process (MDP) could be denoted as $\mathcal{M} := <\mathcal{S}, \mathcal{A}, \mathcal{T}, R>$. 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 $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is calculated to measure the performance improvement in the transition. Those transitions between states and actions achieve a trajectory $\tau := (s_0, a_0, s_1, \cdots, s_T)$. The target of MDP is to find an optimal policy π^* such that the returns (accumulated rewards) in trajectories τ can be maximized:

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

where $\Pi : \mathcal{S} \to \mathcal{A}$ selects an action with a given state, γ 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 π_θ with parameters θ. 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:

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

which is maximized by the gradient ascent with gradient:

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

In this paper, 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:

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

where η 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_ϕ is adopted to estimate the expected return $\hat{G}$ for different states. The loss function of v_ϕ is formulated as:

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

2.2 DE and Its Advanced Variants

As mentioned earlier, this paper uses DE as a case study for our RL-DAS framework. Therefore, in this subsection, 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 [15–17]. Considering the relevance of DE variants to our study, we categorize them into three distinct groups for review.

2.2.1 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 re-selected 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.

2.2.2 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 multi-population 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 multi-population 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 algorithm selection 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 algorithm selection.

2.2.3 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 Reinforcement Learning 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 dynamic algorithm selection, 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.

3 Methodology

3.1 Dynamic Algorithm Selection

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

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

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

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

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

where $\pi : I \times T \to \Lambda$ 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 reinforcement learning methods such as the PPO [13].

3.2 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 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 Λ is extracted and constructed as a state s_t. Based on the state, the policy agent π_θ 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 Γ, an algorithm is warm-started from its last break and applied to envolve 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 Γ 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 π_θ and the critic net v_ϕ. 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 paper, 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 3.3. The neural network design and the training process will be presented in Section 3.4. Last, we introduce the aforementioned algorithm context memory mechanism to support the warm-start of different algorithms in Section 3.5.

3.3 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.

3.3.1 State

The state space of RL-DAS consists of Landscape Analysis (LA) and Algorithm History (AH) information. For the LA part, nine scalar features $(feature_1, \cdots, feature_9)$ are extracted by analyzing the population information and the random walk sampling information on the problem landscape, which forms a 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 $(feature_1, \cdots, feature_9)$, deferring the detailed formulations in Section 1 of our supplementary document.

- $feature_1$: The cost value is a pivotal indicator of optimization status, making it our first feature.

Algorithm 1: Pseudo Code of RL-DAS

Input: Policy π_θ, Critic v_ϕ, Instance Set I, Algorithm Pool Λ with Context Memory Γ
Output: Trained Policy π_θ, Critic v_ϕ
for $epoch \leftarrow 0$ **to** $Epoch$ **do**
   for $instance \in I$ **do**
      Population $P_0 \leftarrow InitPopulation()$;
      Initialize Γ;
      $t \leftarrow 0$;
      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 $< s_t, a_t, s_{t+1}, adc_t, \log \pi_\theta(a_t|s_t) >$;
         $t \leftarrow t + 1$;
         Update Γ;
      end
      Update $adc_{0:T}$ into $r_{0:T}$ by Eq. (13);
      for $k \leftarrow 1$ **to** K **do**
         Update π_θ and v_ϕ by **PPO** method;
      end
   end
end

- $feature_2$: The fitness distance correlation (FDC) [40] describes problem complexity by assessing the relationship between fitness values and solution distances.

- $feature_3$: 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 multi-funnel landscape, this value would be much larger.

- $feature_4$: The maximum distance among population describes the convergence state of population.

- $feature_{5,6,7,8}$: 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.

- $feature_9$: 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 Λ. We maintain two shift vectors for each algorithm to characterize the population distribution change incurred by it. Specifically, let $SV_{\text{best}}^l \in \mathbb{R}^D$ and $SV_{\text{worst}}^l \in \mathbb{R}^D$ denote the average shift of the best and the worst individual caused by the algorithm Λ^l in the previous optimization steps. For a time interval i, if Λ^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 ith step, $sv_{i,\text{best}}^l \in \mathbb{R}^D$, is defined as:

$$sv_{i,\text{best}}^l = x'_{\text{best}} - x_{\text{best}} \tag{8}$$

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

$$SV_{\text{best}}^l = \frac{1}{H^l} \sum_{i=1}^{H^l} sv_{i,\text{best}}^l \tag{9}$$

where H^l is the number of intervals that the algorithm Λ^l is selected. The calculation of SV_{worst}^l is similar. Notably, in the case that the algorithm Λ^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.

$$f_{\text{AH}} = \left[\{SV_{\text{best}}^1, SV_{\text{worst}}^1\} \cdots \{SV_{\text{best}}^L, SV_{\text{worst}}^L\} \right] \tag{10}$$

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

$$state = \{f_{\text{LA}} \in \mathbb{R}^9, f_{\text{AH}} \in \mathbb{R}^{2L \times D}\} \tag{11}$$

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.
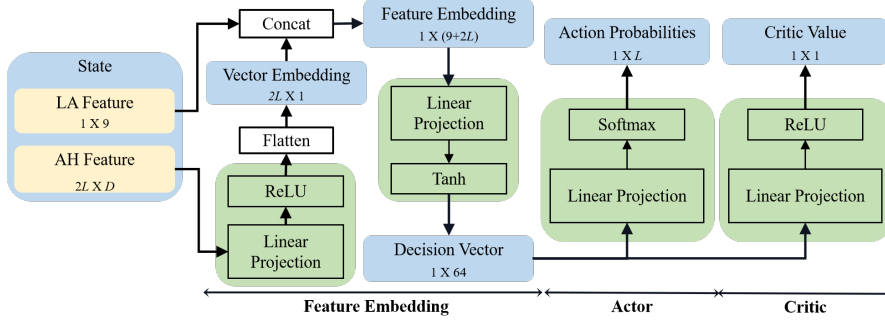
Figure 2: The Neural Network workflow for π_θ (Actor) and v_ϕ (Critic).

3.3.2 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 $a \in [1, L]$.

3.3.3 Reward

In order to guide the agent towards achieving a lower evaluation cost, the reward function should consider the absolute cost reduction at each time step t:

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

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

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

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.

3.4 Network Design

As shown in Fig. 2, the network is composed of three modules: Feature Embedding, Actor, and 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.

3.4.1 Feature Embedding

The Feature Embedding module fuses the f_{LA} and the f_{AH}. First, we feed the f_{AH} into a two-layer feed forward network to generate a more compact embedding vector V_{AH} as

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

where $W_{\text{ve}}^{(1)} \in \mathbb{R}^{D \times 64}, W_{\text{ve}}^{(2)} \in \mathbb{R}^{64 \times 1}$ are two network layers and the activation function σ 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 9-dimensional f_{LA}. The two types of features are concatenated and then fused by another network layer to generate the decision vector representation as

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

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

3.4.2 Actor

Actor decides the probability for selecting an algorithm from the candidate algorithm pool. It first maps DV in Eq. (15) to a logits vector through a two-layer feed-forward 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:

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

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

3.4.3 Critic

Critic feeds DV into a two-layer feed forward network to obtain the estimated return value of the state.

$$b = \sigma(W_{\text{critic}}^{(2)}\sigma(DV^{\top}W_{\text{critic}}^{(1)} + b_{\text{critic}}^{(1)}) + b_{\text{critic}}^{(2)}) \tag{17}$$

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

3.5 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 paper 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:

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

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 Λ_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 Λ_i. This restoration method can be generalized to various algorithms, as it does not propose any special requirements on the contexts.

4 RL-DAS on DE: A 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 paper, 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 2.2, 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.

4.1 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 2.2, which have different strengths and weaknesses in terms of balancing the exploration and exploitation of the population. According to our experiments (which will be discussion 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.

4.2 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 $\Delta \in \left[\frac{MaxFEs}{T_{\max}}, \frac{MaxFEs}{T_{\min}}\right]$. In essence, at every Δ 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 Δ. In Section 5.5, we will conduct a grid experiment to investigate interval lengths that support good performance with acceptable training efficiency.

4.3 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 as the probabilities of adaptive mutation and crossover (τ_1 and τ_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
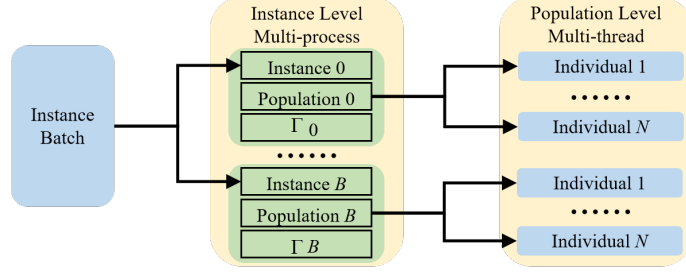
Figure 3: The structure of the hierarchical parallel mechanism. B indicates the batch size.

MadDE and NL-SHADE-RSP, as well as their elite archive ($Arch$) are stored in the $Common$. To summarize, the complete context memory Γ for the three algorithms is organized in the following form:

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

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 Γ are replaced with those updated p_m.

4.4 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, 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 sub-processes. The agent processes their features in batch and returns the actions to their corresponding sub-processes, which determines the next algorithm to use on each instance. Note that the populations and the algorithm context memories are not shared among sub-processes. Pertaining to the population level, we design operators to process the entire population at once using the multi-thread parallel method from Numpy, instead of handling individuals one by one.

5 Experiment

Our experiments study the following research questions:

- 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 5.3.
- RQ2: How well does RL-DAS generalize to unseen problem instances? See Section 5.4.
- RQ3: What are the suggested configurations for RL-DAS? See Section 5.5.

5.1 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 1 unimodal problem (F1), 3 basic problems (F2 - F4), 3 Hybrid problems (F5 - F7) and 3 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 paper 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.

5.2 Competitors

In this paper, 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

Table 2: Comparing RL-DAS with comparison algorithms on Shifted & Rotated 10D 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%**	**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	
		Rand-DAS	100.0% ≈	82.87% −	95.40% −	100.0% ≈	100.0% ≈	99.33% ≈	100.0% ≈	79.87% −	89.91% −	88.52% −	93.69% −	7 / 4 / 0
			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 −	
		MadDE	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 −	1.920e5 −	1.920e5 −	1.878e5 +	
		NL-SHADE-RSP	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% ≈	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% ≈	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% −	87.03% −	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% −	88.77% −	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 3: Comparing RL-DAS with comparison algorithms on Shifted & 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	
		Rand-DAS	100.0% ≈	73.70% −	96.83% ≈	100.0% ≈	100.0% ≈	96.51% −	100.0% ≈	79.14% −	87.03% −	89.48% −	92.35% −	7 / 4 / 0
			6.856e5 −	1.000e6 ≈	1.000e6 ≈	1.000e6 ≈	1.000e6 ≈	1.000e6 ≈	1.000e6 ≈	9.653e5 +	9.456e5 −	9.520e5 −	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 −	

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.

5.3 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 paper, we report the cost decent in percentage, and we additionally provide the absolute cost values in Section 4 of our supplementary document.

5.3.1 Comparison with Backbones

Table 2 and Table 3 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 '≈' 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 '≈'); and the third value counts the cases otherwise. From the results in these tables, we can conclude that:

- 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.

- 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 towards 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

Table 4: 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 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 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 an DRL agent to dynamically switch these algorithms to achieve comprehensively better performance.

- For the 20D problems shown in Table 3, 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 hand-crafted 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.

5.3.2 Comparison between DAS and AS

The DAS and backbones parts of Table 2 and Table 3 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 Algorithm Selection (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:

$$\text{AS}^*(i) = \max(\text{JDE21}(i), \text{MadDE}(i), \text{NL-SHADE-RSP}(i)) \tag{20}$$

where $\text{JDE21}(i)$ 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 Table 2 and Table 3.

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 performance and breaks through the single best candidate algorithm. The outstanding performance of Rand-DAS in Table 2 and Table 3 prove its potential. The employment of an DRL agent ensures wise selection thus RL-DAS defeats AS* significantly.

5.3.3 Comparison with Other Advanced DE Variants

The last part in Table 2 and Table 3 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.

5.3.4 Runtime Comparison

Table 4 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 99-dimensional state representation and one-by-one operator selection for each individual. RL-DAS, on the other hand, completes optimization in 1.434 seconds 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.

5.4 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 5.3: the unimodal problem C1 (Bent Cigar), the basic multimodal problem C2 (Schwefel) and the mix-class problems C11, and then zero-shot transfer them to the other CEC problems. The results are shown in Table 5. 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.

Table 5: 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% ≈	1.046e5 +	100.0% ≈	1.215e5 −	100.0%	1.127e5
C2	84.32% −	**2.000e5** ≈	\	\	83.69% −	**2.000e5** ≈	**85.97%**	**2.000e5**
C3	95.67% ≈	**2.000e5** ≈	95.61% ≈	**2.000e5** ≈	95.86% +	**2.000e5** ≈	95.63%	**2.000e5**
C4	**100.0%** ≈	1.999e5 ≈	**100.0%** ≈	**1.996e5** +	**100.0%** ≈	2.000e5 ≈	**100.0%**	2.000e5
C5	**100.0%** ≈	1.996e5 ≈	**100.0%** ≈	**1.986e5** +	**100.0%** ≈	1.997e5 ≈	**100.0%**	1.996e5
C6	99.37% ≈	**2.000e5** ≈	99.56% +	**2.000e5** ≈	99.38% ≈	**2.000e5** ≈	99.39%	**2.000e5**
C7	**100.0%** ≈	**2.000e5** ≈	**100.0%** ≈	**2.000e5** ≈	**100.0%** ≈	**2.000e5** ≈	**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%** ≈	**1.909e5** +	88.59% −	1.921e5 −	88.79%	1.914e5
	2 / 5 / 2		2 / 2 / 5		5 / 4 / 1		- / - / -	

Table 6: 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		- / - / -	

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 6, where the three backbone algorithms serve as references. RL-DAS outperforms the baselines on most of transfer settings in Table 6. 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.

5.5 Ablation Study and Hyperparameter Research

5.5.1 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 re-initialized at the beginning of each internal.

Table 7 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

5.5.2 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 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%).

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

11

Table 7: The 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%** ≈	1.425e5 −	**100.0%** ≈	1.383e5 −	99.96% −	2.000e5 −	**100.0%**	**1.127e5**
C2	82.29% −	**2.000e5** ≈	82.73% −	**2.000e5** ≈	56.34% −	**2.000e5** ≈	**85.97%**	**2.000e5**
C3	95.31% −	**2.000e5** ≈	95.36% −	**2.000e5** ≈	90.17% −	**2.000e5** ≈	**95.63%**	**2.000e5**
C4	**100.0%** ≈	**2.000e5** ≈	**100.0%** ≈	**2.000e5** ≈	99.96% −	**2.000e5** ≈	**100.0%**	**2.000e5**
C5	**100.0%** ≈	2.000e5 −	**100.0%** ≈	2.000e5 −	99.97% −	2.000e5 −	**100.0%**	**1.996e5**
C6	98.96% −	**2.000e5** ≈	98.91% −	**2.000e5** ≈	95.08% −	**2.000e5** ≈	**99.39%**	**2.000e5**
C7	99.99% −	**2.000e5** ≈	**100.0%** ≈	**2.000e5** ≈	99.21% −	**2.000e5** ≈	**100.0%**	**2.000e5**
C8	79.32% −	1.851e5 −	79.77% −	1.873e5 −	68.43% −	2.000e5 −	**80.48%**	**1.821e5**
C9	89.52% −	1.905e5 −	89.86% −	1.912e5 −	80.28% −	2.000e5 −	**90.22%**	**1.882e5**
C10	88.24% −	1.916e5 ≈	87.76% −	1.933e5 −	77.67% −	2.000e5 −	**88.79%**	**1.914e5**
C11	92.95% −	1.939e5 −	93.16% −	1.948e5 −	84.27% −	2.000e5 −	**93.94%**	**1.903e5**
	10 / 1 / 0		9 / 2 / 0		11 / 0 / 0		- / - / -	

Table 8: The performance under different reward schemes.

Problem Class	Reward Type				
	r1	r2	r3	r4	Ours
C1	**100.0%** ≈	**100.0%** ≈	**100.0%** ≈	**100.0%** ≈	**100.0%**
	1.345e5 −	**1.035e5** +	1.360e5 −	1.359e5 −	1.127e5
C2	85.79% −	85.73% −	85.46% −	83.96% −	**85.97%**
	2.000e5 ≈	**2.000e5** ≈	**2.000e5** ≈	**2.000e5** ≈	**2.000e5**
C3	95.36% −	95.37% −	95.33% −	94.14% −	**95.63%**
	2.000e5 ≈	**2.000e5** ≈	**2.000e5** ≈	**2.000e5** ≈	**2.000e5**
C4	**100.0%** ≈	**100.0%** ≈	**100.0%** ≈	**100.0%** ≈	**100.0%**
	2.000e5 ≈	**2.000e5** ≈	**2.000e5** ≈	**2.000e5** ≈	**2.000e5**
C5	**100.0%** ≈	**100.0%** ≈	**100.0%** ≈	**100.0%** ≈	**100.0%**
	2.000e5 −	1.999e5 −	2.000e5 −	2.000e5 −	**1.996e5**
C6	99.01% −	98.46% −	98.68% −	97.91% −	**99.39%**
	2.000e5 ≈	**2.000e5** ≈	**2.000e5** ≈	**2.000e5** ≈	**2.000e5**
C7	**100.0%** ≈	**100.0%** ≈	**100.0%** ≈	**100.0%** ≈	**100.0%**
	2.000e5 ≈	**2.000e5** ≈	**2.000e5** ≈	**2.000e5** ≈	**2.000e5**
C8	80.36% ≈	80.01% −	79.77% −	80.43% ≈	**80.48%**
	1.923e5 −	**1.798e5** +	1.891e5 −	1.957e5 −	1.821e5
C9	**90.20%** ≈	89.86% −	89.73% −	89.17% −	**90.22%**
	1.924e5 −	**1.880e5** ≈	1.916e5 −	1.943e5 −	1.882e5
C10	88.61% −	88.55% −	88.16% −	87.63% −	**88.79%**
	1.951e5 −	1.916e5 ≈	1.943e5 −	1.949e5 −	**1.914e5**
C11	93.65% −	93.56% −	93.52% −	93.07% −	**93.94%**
	1.924e5 −	**1.886e5** +	1.921e5 −	1.941e5 −	1.903e5
	9 / 2 / 0	8 / 2 / 1	9 / 2 / 0	9 / 2 / 0	- / - / -

Figure 4: The performance change along schedule intervals.

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

5.5.3 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 show 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 paper for RL-DAS.

6 Conclusion

This paper 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.

APPENDIX

A   Detailed Definition of LA Features

In the main body of our paper, we briefly introduce the main idea of the 9 LA features. In this section, we present their detailed formulation:

a) $feature_1$: For the first feature measuring the value of cost, we adopt the current best cost $cost_t^*$ normalized by the best cost $cost_0^*$ in the initial population as an optimization progress feature:

$$feature_1 = \frac{cost_t^*}{cost_0^*} \tag{25}$$

b) $feature_2$: The fitness distance correlation (FDC) [40] is a feature that describes the complexity of the problem by evaluating the relationship between fitness value and the distance of the solution from the optimum. Given a set of individuals' distance to the best individual $Dist = \{d_1^*, \cdots, d_N^*\}$ and their costs $C = \{c_1, \cdots, c_N\}$, the feature is calculated as:

$$feature_2 = \frac{\frac{1}{N}\sum_{i=1}^{N}(c_i - \bar{c})(d_i^* - \bar{d}^*)}{\delta_{Dist}\delta_C} \tag{26}$$

where δ_{Dist} and δ_C are the variances of $Dist$ and C.

c) $feature_3$: The dispersion difference [41] measures the distribution difference between the top 10% and the whole population:

$$feature_3 = \bar{d}_{\text{top}} - \bar{d} \tag{27}$$

where $\bar{d}_{\text{top}}$ is the average pairwise distance of top 10% individuals. 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 multi-funnel landscape, this value would be much larger.

d) $feature_4$: The maximum distance among population describes the convergence state of population. It is calculated by the maximum distance in the population and normalized by the diameter of the search space:

$$feature_4 = \frac{d_{\text{max}}}{diameter} \tag{28}$$

e) $feature_5$: The next four features are adopted to measure the evolvability of the population on the given problem. First, the negative slope coefficient (NSC) [42] measures the difference between the costs change incurred by the population evolution and by making small tentative random walk steps. In this paper, these tentative steps are made by two randomly selected algorithms from Λ. Given the ascending sorted evolved population cost set (C) and the randomly sampled population cost set (C') that are both uniformly segmented into m parts as $\{C_1, \cdots, C_m\}$ and $\{C_1', \cdots, C_m'\}$, NSC is computed by:

$$feature_5 = \min\left(\sum_{i=1}^{m}\frac{\bar{C}_{i+1} - \bar{C}_i}{\bar{C}'_{i+1} - \bar{C}'_i}, 0\right) \tag{29}$$

which reflects the hardness in solving the problem: a high value such as 0 indicates an easy problem, whereas a negative NSC indicates a more complex problem landscape.

f) $feature_6$: The average neutral ratio (ANR) [43] has a similar consideration of NSC, but it measures the performance changes between the current population and S different populations sampled through candidate algorithms. Given the cost set of sampled populations $\{C'^1, \cdots, C'^S\}$ where $C'^j = \{c_1^j, \cdots, c_N^j\}$, this feature is calculated as

$$feature_6 = \frac{\sum_i^N \sum_j^S |c_i - c_i^j| < eps}{N \cdot S} \tag{30}$$

where eps is a precision value. The higher the ANR is, the problem landscape is more rugged.

g) $feature_7$: Inspired by Wang et al. [44], we also propose the Best-Worst Improvement to measure the evolvability. The Best Improvement ($feature_7$) measures the optimization difficulty of the current population, which is calculated by the ratio of sampled individuals that make no improvement over the current population:

$$feature_7 = \frac{\sum_i^N \alpha_i}{N} \tag{31}$$

where

$$\alpha_i = \begin{cases} 0 & \text{if } \left(\sum_j^S \mathbb{I}(c_i^j < c_i)\right) > 0 \\ 1 & \text{otherwise} \end{cases} \tag{32}$$

Here, α_i is the contribution of the i-th individual in the current population (c_i is its cost), and c_i^j is the cost of the i-th individual in the j-th sampled population. The α_i will be 1 if the current individual i is the best over the sampled individuals with small potential to be improved.

h) $feature_8$: Contrary to the Best Improvement, the Worst Improvement ($feature_8$) describes the optimization potential and uses the ratio of individuals that do not get worse:

$$feature_8 = \frac{\sum_i^N \beta_i}{N} \tag{33}$$

where

$$\beta_i = \begin{cases} 0 & \text{if } \left(\sum_j^S \mathbb{I}(c_i^j > c_i)\right) < S \\ 1 & \text{otherwise} \end{cases} \tag{34}$$

Here, β_i will be 1 if the individual is the worst among the sampled individuals with high potential.

Taxonomy		Algorithm		Problem Class										
				C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11
DAS		**RL-DAS**	Mean	**2.974e-8**	**3.334e2**	**1.738e1**	9.982e-1	**1.587e1**	**5.659e0**	**9.783e0**	**5.192e1**	**7.988e1**	1.715e2	**7.386e1**
			Std	**2.156e-8**	**1.108e2**	**1.666e0**	3.487e-1	**2.314e1**	**3.211e0**	**1.058e1**	**1.259e1**	**2.468e1**	1.501e1	**2.364e1**
		Rand-DAS	Mean	6.473e-8	4.071e2	1.828e1	1.113e0	1.754e1	6.216e0	9.997e0	5.355e1	8.242e1	1.757e2	7.691e1
			Std	3.468e-7	1.546e2	2.342e0	4.679e-1	3.178e1	4.156e0	1.189e1	1.673e1	3.479e1	1.712e1	2.856e1
AS		AS*	Mean	5.424e-8	3.415e2	1.833e1	**9.546e-1**	1.628e1	6.123e0	9.817e0	5.357e1	8.217e1	1.752e2	7.668e1
			Std	1.867e-9	1.068e2	2.153e0	**3.681e-1**	2.610e1	3.798e0	9.998e0	1.351e1	2.754e1	1.216e1	2.234e1
Backbones		JDE21	Mean	8.675e-7	6.292e2	2.005e1	1.102e0	1.273e2	9.185e0	2.918e1	5.650e1	9.418e1	1.771e2	9.727e1
			Std	1.122e-6	2.531e2	2.531e0	3.648e-1	2.354e2	5.254e0	3.047e1	1.276e1	2.317e1	1.056e1	3.179e1
		MadDE	Mean	4.162e-8	3.888e2	1.834e1	9.757e-1	1.834e1	6.123e0	1.018e1	5.578e1	8.356e1	1.847e2	7.667e1
			Std	4.311e-8	1.135e2	2.649e0	4.17e-1	2.228e1	3.105e0	1.086e1	1.233e1	3.670e1	1.168e1	2.305e1
		NL-SHADE-RSP	Mean	7.892e-4	4.577e2	1.901e1	9.810e-1	1.781e1	2.097e1	1.135e1	5.477e1	8.258e1	1.766e2	8.703e1
			Std	8.697e-3	1.356e2	2.462e0	3.471e-1	2.657e1	1.348e1	1.218e1	1.576e1	3.971e1	1.210e1	2.897e1
Other Advanced DE Variants	Adaptive	EPSDE	Mean	1.789e-7	5.767e2	1.780e1	1.072e0	2.047e1	7.144e0	1.105e1	6.267e1	1.045e2	1.502e2	9.227e1
			Std	2.456e-7	2.015e2	2.049e0	4.077e-1	3.542e1	4.351e0	1.089e0	1.437e1	3.688e1	9.547e0	3.151e1
	Ensemble	EDEV	Mean	2.015e-7	5.358e2	1.778e1	9.954e-1	2.258e1	6.772e0	1.096e1	6.020e1	1.009e2	**1.368e2**	8.508e1
			Std	2.346e-7	1.713e2	1.941e0	3.178e-1	2.985e1	3.789e0	1.131e1	1.476e1	2.977e1	**9.471e0**	2.777e1
	RL Assisted	DEDQN	Mean	6.492e3	6.737e2	3.011e1	1.036e0	1.366e2	1.203e2	2.987e1	6.360e1	1.354e2	1.859e2	1.053e2
			Std	7.279e3	2.346e2	3.468e0	3.017e-1	1.791e2	5.671e1	3.154e1	1.279e1	2.057e1	1.636e1	3.781e1
		DEDDQN	Mean	5.271e3	6.658e2	2.701e0	1.076e0	1.321e2	1.041e2	2.541e1	6.158e1	1.264e2	1.826e2	1.065e2
			Std	6.342e3	1.946e2	2.978e0	2.814e-1	2.457e2	5.547e1	2.454e1	1.219e1	2.153e1	1.713e1	3.687e1
		LDE	Mean	2.152e1	5.598e2	1.833e1	9.897e-1	1.835e1	1.011e1	1.215e1	6.057e1	1.328e2	1.793e2	9.507e1
			Std	6.153e1	1.331e2	1.798e0	2.823e-1	2.847e1	5.133e0	1.200e1	1.768e1	3.336e1	1.193e1	3.044e1

Table 10: Comparing RL-DAS with comparison algorithms on Shifted & Rotated 20D problems.

Taxonomy		Algorithm		Problem Class										
				C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11
DAS		**RL-DAS**	Mean	**2.289e-8**	**1.385e3**	**4.392e1**	**2.034e0**	**2.012e2**	**5.342e1**	**6.079e1**	**5.631e1**	**1.333e2**	3.102e2	**2.302e2**
			Std	**6.357e-8**	**3.482e2**	**5.346e0**	**8.548e-1**	**5.841e1**	**2.104e1**	**3.159e1**	**3.111e1**	**4.512e1**	4.898e1	**6.101e1**
		Rand-DAS	Mean	1.087e-3	1.553e3	4.477e1	2.623e0	2.473e2	8.921e1	9.983e1	5.756e1	1.388e2	3.180e2	2.470e2
			Std	1.247e-3	3.678e2	5.752e0	1.504e0	9.493e1	5.189e1	5.498e1	4.199e1	5.647e1	6.677e1	7.122e1
AS		AS*	Mean	7.786e-8	1.425e3	4.957e1	2.298e0	2.166e2	6.083e1	6.584e1	5.833e1	1.378e2	3.207e2	2.399e2
			Std	6.871e-8	2.955e2	5.014e0	1.114e0	7.112e1	2.548e1	3.088e1	3.657e1	3.044e1	3.664e1	6.155e1
Backbones		JDE21	Mean	8.903e-1	2.201e3	5.310e1	4.627e0	3.539e2	1.265e2	1.463e2	6.147e1	1.440e2	3.229e2	2.951e2
			Std	9.368e-1	4.158e2	4.955e0	2.015e0	1.292e2	5.448e1	6.912e1	4.694e1	2.108e1	2.514e1	7.624e1
		MadDE	Mean	8.529e-8	1.431e3	4.900e1	2.551e0	2.333e2	6.083e1	7.094e1	6.012e1	1.428e2	3.316e2	2.376e2
			Std	7.156e-8	2.948e2	4.862e0	1.151e-1	7.745e1	3.161e1	3.318e1	3.991e1	1.983e1	2.167e1	5.595e1
		NL-SHADE-RSP	Mean	5.489e0	1.738e3	6.256e1	7.921e0	3.752e2	1.805e2	1.737e2	6.310e1	1.461e2	3.821e2	2.870e2
			Std	6.017e0	3.481e2	5.515e0	3.100e0	1.519e2	8.575e1	8.321e1	4.561e1	1.686e1	3.818e1	7.177e1
Other Advanced DE Variants	Adaptive	EPSDE	Mean	4.320e-5	2.107e3	5.112e1	3.630e0	3.647e2	7.055e1	1.147e2	5.841e1	1.450e2	3.038e2	2.738e2
			Std	4.677e-5	3.982e2	4.844e0	2.554e0	1.223e2	3.149e1	5.516e1	3.694e1	3.419e1	3.422e1	6.984e1
	Ensemble	EDEV	Mean	1.087e-5	1.998e3	5.041e1	3.144e0	3.580e2	6.850e1	1.205e2	5.869e1	1.439e2	**3.032e2**	2.796e2
			Std	1.583e-5	3.492e2	5.344e0	1.949e0	1.323e2	3.240e1	5.794e1	3.741e1	3.947e1	**3.632e1**	6.888e1
	RL Assisted	DEDQN	Mean	7.781e0	2.141e3	6.524e1	5.465e0	5.713e2	9.943e1	1.780e2	6.856e1	1.549e2	3.625e2	2.938e2
			Std	8.975e0	3.833e2	7.589e0	3.697e0	2.642e2	5.947e1	1.009e2	4.198e1	7.165e1	7.548e1	7.171e1
		DEDDQN	Mean	6.513e0	2.232e3	6.849e1	4.874e0	5.551e2	8.844e1	1.686e2	6.647e1	1.495e2	3.624e2	2.889e2
			Std	8.147e0	4.895e2	8.687e0	3.211e0	2.156e2	4.969e1	9.699e1	3.669e1	6.558e1	7.794e1	7.089e1
		LDE	Mean	6.498e-4	2.289e3	5.705e1	3.211e0	4.301e2	6.288e1	9.654e1	6.305e1	1.466e2	3.471e2	2.799e2
			Std	6.954e-4	3.161e2	6.939e0	1.941e0	1.982e2	4.177e1	5.900e1	3.766e1	7.556e1	6.999e1	7.184e1

i) $feature_9$: It is beneficial to let the agent be aware of the consumption of the computational budget, so we introduce the ratio of consumed FEs over the maximum as a feature:

$$feature_9 = \frac{FEs}{MaxFEs} \tag{35}$$

B Experiment Settings

The parameters of JDE21, MadDE, and NL-SHADE-RSP integrated in RL-DAS are kept the same as in their original papers. The exceptions are the parameters shared across the candidates and have different settings in different algorithms, such as the size of elite archive, parameter memories and population. To fix these gaps, we choose the maximal settings $2.3 \times N$ for the archive size (following MadDE) and $20 \times D$ for the memory sizes (following NL-SHADE-RSP). The population size range follows the setting of JDE21 to ensure its multi-population division, which is [30, 170]. The parameters for comparisons are kept the same as in their original papers.

The $MaxFEs$ is set to 200,000 for 10D problems and 1,000,000 for 20D problems following the conventions of CEC2021 competition (note that the FEs costed in the fitness landscape analysis for state representation of RL-DAS are considered for a fair comparison). The schedule interval Δ is set to 2,500 for 10D and 8,000 for 20D problems. During training, the instance sets are batched in the size of 16. The agent learns for 200 epochs with the learning rate $\lambda = 10^{-5}$ for both of the actor and the critic. For the testing, each algorithm is executed 30 runs for each test instance. The algorithm is terminated when either the $MaxFEs$ is exhausted or the best cost is lower than the termination error 10^{-8}.

C In-Depth Analysis of DAS

This study uses DRL to learn a policy that dynamically selects the most proper EC algorithm according to the optimization state. The experimental results in the main body of our paper have proven the efficiency of RL-DAS. This subsection conducts an in-depth analysis of the learned policy, by depicting the search behaviors of different candidate algorithms and the selecting results of RL-DAS during the optimization
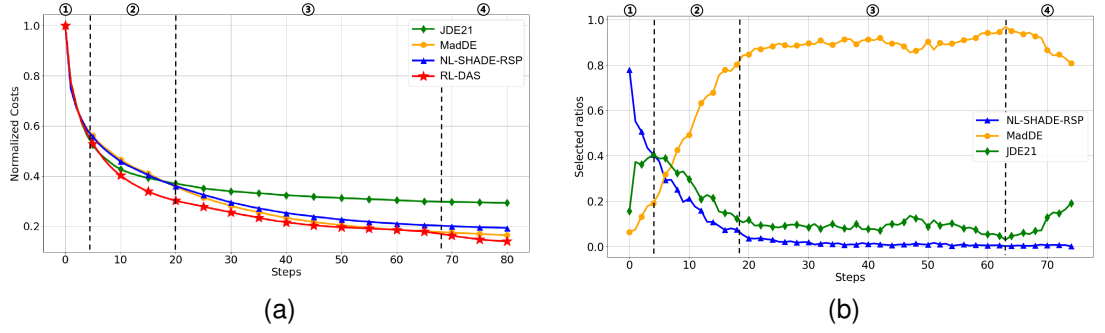
Figure 5: The descent curve (a) and selection distribution (b) on C2.

progress, where the 10D Schwefel problem is taken as an example for illustration. The results are shown in Fig. 5. Specifically, Fig. 5a records the normalized costs obtained by different algorithms for every 2,500 function evaluations, whereas Fig. 5b reports the proportion of selecting each candidate algorithm in each RL step. Note that the abscissa of these two sub-figures cannot be completely aligned: due to additional function evaluations consumed by random walk sampling in states, the actual evaluations allocated to each RL step is larger than 2,500, therefore the total number of RL steps in Fig. 5b is less than the recorded steps in Fig. 5a.

First, from Fig. 5a, it can be observed that NL-SHADE-RSP searches slightly faster than the JDE21 and MadDE in the initial steps (phase 1). At this time, from Fig. 5b, the selection proportion of NL-SHADE-RSP is close to 80%, indicating that the RL-DAS agent tries to make good use of the best performance of NL-SHADE-RSP at the beginning. But this proportion quickly decreases, while the selection proportion of JDE21 increases. This is because JDE21 quickly surpasses the NL-SHADE-RSP and becomes the best algorithm with its strong exploitation strategy. Then, as JDE21 gradually converges (phase 2), RL-DAS reduces its proportion and turns to the best algorithm, MadDE, after 20 steps (phase 3), to perform the exploratory search. Note that, at this stage, RL-DAS still retains some selections on JDE21 to achieve a balance between exploration and exploitation. This selection preference is kept until the last 10 steps (phase 4), where more JDE21 options are chosen to make a sprint, indicating that the agent realizes that the optimization is about to end and hence final exploitation is required. The performance of RL-DAS in this period has a promotion that makes the final performance of RL-DAS defeat others. The experimental result proves that the agent has learned the different characteristics among candidates and is able to adapt the optimization behavior according to the environment, helping RL-DAS achieve both the fastest convergence and the highest optimization accuracy.

Table 11: 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	Mean	\	**2.058e-8**	6.477e-8	2.974e-8
	Std	\	**2.231e-8**	3.104e-8	2.156e-8
C2	Mean	3.726e2	\	3.876e2	**3.334e2**
	Std	1.339e2	\	1.258e2	**1.108e2**
C3	Mean	1.722e1	1.746e1	**1.647e1**	1.738e1
	Std	1.677e0	1.688e0	**1.587e0**	1.666e0
C4	Mean	1.099e0	**9.971e-1**	1.014e0	9.982e-1
	Std	3.363e-1	**3.746e-1**	4.187e-1	3.487e-1
C5	Mean	1.623e1	1.599e1	1.626e1	**1.587e1**
	Std	2.510e1	2.228e1	2.481e1	**2.314e1**
C6	Mean	5.844e0	**4.082e0**	5.752e0	5.659e0
	Std	3.487e0	**3.014e0**	3.342e0	3.211e0
C7	Mean	9.854e0	**9.777e0**	9.876e0	9.783e0
	Std	1.099e1	**9.627e0**	9.811e0	1.058e1
C8	Mean	**5.152e1**	5.248e1	5.243e1	5.192e1
	Std	**1.164e1**	1.394e1	1.388e1	1.259e1
C9	Mean	8.053e1	8.388e1	8.225e1	**7.988e1**
	Std	2.496e1	2.798e1	2.550e1	**2.468e1**
C10	Mean	1.787e2	**1.712e2**	1.746e2	1.715e2
	Std	1.597e1	**1.581e1**	1.670e1	1.501e1

Table 12: Transfer results under different problem class partitions.

Transfer Setting		NL-SHADE -RSP	MadDE	JDE21	RL-DAS (Zero-shot)
TS1	Mean	7.018e1	4.457e1	6.118e1	**4.312e1**
	Std	2.369e1	1.117e1	1.569e1	**1.144e1**
TS2	Mean	8.900e1	6.198e1	8.170e1	**6.110e1**
	Std	2.960e1	1.321e1	2.589e1	**1.296e1**
TS3	Mean	1.349e2	1.245e2	1.334e2	**1.219e2**
	Std	2.518e1	2.347e1	2.499e1	**1.931e1**

D Absolute Cost Values

In the main body of our paper, we report the cost decent in percentage of different algorithms from the random initialization. This section additionally presents the detailed absolute results of the algorithms, where the mean and standard deviations are presented. Specifically, Table 9 and Table 10 supplement the results for major performance comparisons of RL-DAS and the competitors. Table 11 and Table 12 shows the

Table 13: The ablation study performance.

Problem Class		RL-DAS w/o LA	RL-DAS w/o AH	RL-DAS w/o Context	RL-DAS
C1	Mean	6.872e-6	8.931e-7	4.587e6	**2.974e-8**
	Std	7.361e-6	1.116e-6	4.697e6	**2.156e-8**
C2	Mean	4.208e2	4.104e2	1.038e3	**3.334e2**
	Std	1.441e2	1.534e2	3.697e2	**1.108e2**
C3	Mean	1.865e1	1.845e1	3.909e1	**1.738e1**
	Std	1.838e0	1.748e0	5.201e0	**1.666e0**
C4	Mean	1.157e0	1.143e0	2.698e2	**9.982e-1**
	Std	5.069e-1	4.987e-1	9.697e1	**3.487e-1**
C5	Mean	1.603e1	1.618e1	6.972e2	**1.587e1**
	Std	2.669e1	2.589e1	9.676e2	**2.314e1**
C6	Mean	9.648e0	1.011e1	4.564e1	**5.659e0**
	Std	6.033e0	6.352e0	2.112e1	**3.211e0**
C7	Mean	3.145e1	1.593e1	2.331e2	**9.783e0**
	Std	3.672e1	1.649e1	2.522e2	**1.058e1**
C8	Mean	5.500e1	5.381e1	8.397e1	**5.192e1**
	Std	1.415e1	1.399e1	2.690e1	**1.259e1**
C9	Mean	8.560e1	8.282e1	1.611e2	**7.988e1**
	Std	2.971e1	2.566e1	5.998e1	**2.468e1**
C10	Mean	1.799e2	1.873e2	3.416e2	**1.715e2**
	Std	1.680e1	1.659e1	4.644e1	**1.501e1**
C11	Mean	8.592e1	8.337e1	1.917e2	**7.386e1**
	Std	3.786e1	3.668e1	7.250e1	**2.364e1**

Table 14: The optimization performance under different reward schemes.

Problem Class		Reward Type				
		r1	r2	r3	r4	Ours
C1	Mean	6.894e-8	**2.182e-8**	7.565e-8	7.553e-8	2.974e-8
	Std	6.289e-8	**1.837e-8**	7.462e-8	7.361e-8	2.156e-8
C2	Mean	3.377e2	3.391e2	3.455e2	3.812e2	**3.334e2**
	Std	1.298e2	1.295e2	1.259e2	1.441e2	**1.108e2**
C3	Mean	1.845e1	1.841e1	1.857e1	2.331e1	**1.738e1**
	Std	1.737e0	1.716e0	1.677e0	1.898e0	**1.666e0**
C4	Mean	1.116e0	1.088e0	1.171e0	1.128e0	**9.982e-1**
	Std	4.195e-1	3.866e-1	4.359e-1	4.514e-1	**3.487e-1**
C5	Mean	1.777e1	1.690e1	1.797e1	1.861e1	**1.587e1**
	Std	2.311e1	2.396e1	2.557e1	2.564e1	**2.314e1**
C6	Mean	9.184e0	1.429e1	1.225e1	1.939e1	**5.659e0**
	Std	5.414e0	8.333e0	6.122e0	9.935e0	**3.211e0**
C7	Mean	1.132e1	9.852e0	1.006e1	1.102e1	**9.783e0**
	Std	1.224e1	1.069e1	1.103e1	1.206e1	**1.058e1**
C8	Mean	5.224e1	5.317e1	5.381e1	5.205e1	**5.192e1**
	Std	1.219e1	1.314e1	1.355e1	1.268e1	**1.259e1**
C9	Mean	8.004e1	8.282e1	8.388e1	8.846e1	**7.988e1**
	Std	2.317e1	2.647e1	2.591e1	2.970e1	**2.468e1**
C10	Mean	1.742e2	1.752e2	1.811e2	1.892e2	**1.715e2**
	Std	1.584e1	1.543e1	1.682e1	1.675e1	**1.501e1**
C11	Mean	7.740e1	7.849e1	7.898e1	8.446e1	**7.386e1**
	Std	2.410e1	2.938e1	2.863e1	3.217e1	**2.364e1**

detailed results of the zero-shot transfer experiment. Table 13 is the results for the ablation study on framework components, while Table 14 delves into the results of the reward study.

References

[1] A. Slowik and H. Kwasnicka, "Evolutionary algorithms and their applications to engineering problems," *Neural Computing and Applications*, vol. 32, no. 16, pp. 12 363–12 379, 2020.

[2] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[3] R. Storn and K. Price, "Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, p. 341, 1997.

[4] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948.

[5] D. H. Wolpert, W. G. Macready *et al.*, "No free lunch theorems for search," Citeseer, Tech. Rep., 1995.

[6] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann, "Automated algorithm selection: Survey and perspectives," *Evolutionary Computation*, 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," *Artificial Intelligence*, vol. 126, no. 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 *Principles and Practice of Constraint Programming-CP 2009: 15th International Conference, CP 2009 Lisbon, Portugal, September 20-24, 2009 Proceedings 15*, 2009, pp. 142–157.

[10] K. Xue, J. Xu, L. Yuan, M. Li, C. Qian, Z. Zhang, and Y. Yu, "Multi-agent dynamic algorithm configuration," in *Advances in Neural Information Processing Systems*, 2022.

[11] A. Biedenkapp, H. F. Bozkurt, T. Eimer, F. Hutter, and M. Lindauer, "Dynamic algorithm configuration: Foundation of a new meta-algorithmic framework," in *ECAI 2020*. IOS Press, 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 *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, 2012, pp. 313–320.

[13] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[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," in *Tech. Rep.* Singapore: Nanyang Technological University, November 2020.

[15] W. Gong, Y. Wang, Z. Cai, and L. Wang, "Finding multiple roots of nonlinear equation systems via a repulsion-based adaptive differential evolution," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 50, no. 4, pp. 1499–1513, 2018.

[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 Transactions on Systems, Man, and Cybernetics: Systems*, vol. 51, no. 5, pp. 2712–2725, 2019.

[17] W. Deng, J. Xu, X.-Z. Gao, and H. Zhao, "An enhanced msiqde algorithm with novel multiple strategies for global optimization problems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 52, no. 3, pp. 1578–1587, 2020.

[18] R. Mallipeddi, P. N. Suganthan, Q.-K. Pan, and M. F. Tasgetiren, "Differential evolution algorithm with ensemble of parameters and mutation strategies," *Applied Soft Computing*, 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," *Information Sciences*, vol. 622, pp. 235–268, 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 Transactions on Evolutionary Computation*, vol. 10, no. 6, pp. 646–657, 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 *2021 IEEE Congress on Evolutionary Computation (CEC)*, 2021, pp. 817–824.

[22] R. Tanabe and A. Fukunaga, "Success-history based parameter adaptation for differential evolution," in *2013 IEEE Congress on Evolutionary Computation*, 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 *2021 IEEE Congress on Evolutionary Computation (CEC)*, 2021, pp. 809–816.

[24] ——, "Nl-shade-lbc algorithm with linear parameter adaptation bias change for cec 2022 numerical optimization," in *2022 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2022, pp. 01–08.

[25] S. Biswas, D. Saha, S. De, A. D. Cobb, S. Das, and B. A. Jalaian, "Improving differential evolution through bayesian hyperparameter optimization," in *2021 IEEE Congress on Evolutionary Computation (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," *Information Sciences*, vol. 423, pp. 172–186, 2018.

[27] J. Zhang and A. C. Sanderson, "Jade: adaptive differential evolution with optional external archive," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 5, pp. 945–958, 2009.

[28] G. Li, Q. Lin, L. Cui, Z. Du, Z. Liang, J. Chen, N. Lu, and Z. Ming, "A novel hybrid differential evolution algorithm with modified code and jade," *Applied Soft Computing*, vol. 47, pp. 577–599, 2016.

[29] S. Li, W. Gong, L. Wang, and Q. Gu, "Evolutionary multitasking via reinforcement learning," *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2023.

[30] Z. Liao, W. Gong, and S. Li, "Two-stage reinforcement learning-based differential evolution for solving nonlinear equations," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 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 & Intelligent Systems*, pp. 1–33, 2023.

[32] Z. Tan and K. Li, "Differential evolution with mixed mutation strategy based on deep reinforcement learning," *Applied Soft Computing*, vol. 111, p. 107678, 2021.

[33] M. Sharma, A. Komninos, M. López-Ibáñez, and D. Kazakov, "Deep reinforcement learning based parameter control in differential evolution," in *Proceedings of the Genetic and Evolutionary Computation Conference*. Association for Computing Machinery, 2019, pp. 709–717.

[34] C. J. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. -, pp. 279–292, 1992.

[35] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *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 *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, 2016.

[37] J. Sun, X. Liu, T. Bäck, and Z. Xu, "Learning adaptive differential evolution algorithm from optimization experiences by policy gradient," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 4, pp. 666–680, 2021.

[38] A. Graves and A. Graves, "Long short-term memory," *Supervised Sequence Labelling with Recurrent Neural Networks*, 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 and Evolutionary Computation*, vol. 75, no. -, p. 101194, 2022.

[40] M. Tomassini, L. Vanneschi, P. Collard, and M. Clergue, "A study of fitness distance correlation as a difficulty measure in genetic programming," *Evolutionary Computation*, vol. 13, no. 2, pp. 213–239, 2005.

[41] M. Lunacek and D. Whitley, "The dispersion metric and the cma evolution strategy," in *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, 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 *Genetic and Evolutionary Computation–GECCO 2004: Genetic and Evolutionary Computation Conference, Seattle, WA, USA, June 26-30, 2004. Proceedings, Part II*, 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 *Genetic Programming: 10th European Conference, EuroGP 2007, Valencia, Spain, April 11-13, 2007. Proceedings 10*, 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 Transactions on Evolutionary Computation*, vol. 22, no. 4, pp. 550–563, 2017.

[45] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[46] D. Hilbert, E. Schmidt, and E. Schmidt, "Zur theorie der linearen und nichtlinearen integralgleichungen: I. teil: Entwicklung willkürlicher funktionen nach systemen vorgeschriebener," *Integralgleichungen und Gleichungen mit unendlich vielen Unbekannten*, pp. 190–233, 1989.