INFORMATICA, 2026, Vol. 0, No. 0, 1–34

DOI: https://doi.org/10.15388/26-INFOR636

An Effective Differential Evolution Based on Fitness Landscape Analysis for Single-Objective Bound-Constrained Optimization

Chun-Wei TSAI[1,3,*], Cheng-Chun CHEN[1], Kuan-Heng CHEN[1], Rokas GIPIŠKIS[2], Olga KURASOVA[2]

[1] *Department of Computer Science and Engineering, National Sun Yat-sen University, 70 Lien-hai Road, Kaohsiung, Taiwan*
[2] *Institute of Data Science and Digital Technologies, Vilnius University, Akademijos g. 4, LT-08412, Vilnius, Lithuania*
[3] *School of Electronics Engineering, Kyungpook National University, 80 Daehak-ro, Buk-gu, Daegu, Korea*
e-mail: cwtsai@mail.cse.nsysu.edu.tw

Received: April 2026; accepted: June 2026

Abstract. Considering the landscape information of the optimization problem represents a promising research direction, as landscape features can provide a metaheuristic algorithm with useful information for understanding the search state, allowing it to adjust the search strategy accordingly. As a successful branch of metaheuristic algorithms for solving single-objective bound-constrained problems, however, the original design of differential evolution (DE) does not sufficiently consider information about the landscape of the solution space. To address this, an effective DE based on reinforcement learning (RL) that dynamically selects a suitable mutation operator via landscape information during the convergence process is proposed in this study, which consists of two phases: offline and online. In the "offline" phase, the proposed algorithm uses an RL-based algorithm to construct a learning model to understand the relationships between landscape characteristics and search operators. Then, in the "online" phase, the learning model constructed from the offline phase and a lightweight fitness landscape analysis (FLA) method are used by the DE to dynamically determine the suitable mutation operator based on the search state encountered by DE during the convergence process, which can reduce the costs for the FLA in every iteration. To understand the performance of the proposed algorithm, the CEC2021 and CEC2022 benchmark functions are used to evaluate its search performance against different DE-based algorithms for solving single-objective optimization problems. Simulation results show that the proposed algorithm outperforms other state-of-the-art DE-based algorithms and other DE algorithms based on FLA and RL in most cases.

Key words: single-objective optimization, differential evolution, fitness landscape analysis, reinforcement learning, and metaheuristic algorithm.

*Corresponding author.

1. Introduction

Metaheuristic algorithms have demonstrated their potential in a large number of studies (Glover and Kochenberger, 2003; Blum and Roli, 2003; Dokeroglu *et al.*, 2019; Tsai and Chiang, 2023) and applications (Soler-Dominguez *et al.*, 2017; Osaba *et al.*, 2021; Elaziz *et al.*, 2021; Sharma and Tripathi, 2022; Yaqoob *et al.*, 2024). From the perspective of algorithm development, it can be seen that various metaheuristic algorithms, e.g. tabu search (TS) in Glover (1989), simulated annealing in Kirkpatrick *et al.* (1983), genetic algorithm (GA) in Holland (1975), ant colony optimization (ACO) in Dorigo *et al.* (1991), particle swarm optimization (PSO) in Kennedy and Eberhart (1995), and differential evolution (DE) in Storn and Price (1995), have been presented to solve optimization problems for years. Many new metaheuristic algorithms have also been presented in recent years (Dokeroglu *et al.*, 2019; Abualigah *et al.*, 2022; Velasco *et al.*, 2024). From the perspective of applications, several studies show that metaheuristic algorithms can be applied to many real-world applications, such as the internet of things (IoT) device control (Sharma and Tripathi, 2022), stock investments (Soler-Dominguez *et al.*, 2017), and even the automatic design of deep neural network architectures (Elaziz *et al.*, 2021; Liu *et al.*, 2023). In addition to using metaheuristic algorithms for real-world applications, an interesting reference workflow for a metaheuristic algorithm can be found in a recent study (Osaba *et al.*, 2021) that explained how to use a metaheuristic algorithm to solve real-world optimization problems, ranging from problem modelling, algorithm design, and solution encoding, to performance assessment, comparison, and replicability.

Researchers in this field have focused on enhancing search performance by presenting a new metaheuristic algorithm, modifying the transition operator, and combining two (meta)heuristic algorithms (Tsai and Chiang, 2023). It is well established that no metaheuristic algorithm is optimal for all optimization problems (Wolpert and Macready, 1997). However, this does not mean that they are not useful; on the contrary, every metaheuristic algorithm has its pros and cons. For example, the tabu list of TS (Glover, 1989) can be used to avoid searching for the same solution too often. The pheromone table of ACO (Dorigo *et al.*, 1991) is used to accumulate the search results of all searches from the very beginning, while the personal and global best solutions of PSO (Kennedy and Eberhart, 1995) are used to guide the search to promising regions in the solution space. Moreover, in addition to modifying the transition operator and combining two (meta)heuristic algorithms to improve results or accelerate the convergence speed, "analysing the landscape of solution space" (Pitzer and Affenzeller, 2012; Ochoa and Malan, 2019; Zou *et al.*, 2022) and "using a machine learning (ML) algorithm" (Talbi, 2021) to determine a suitable search strategy during the convergence process are two promising research directions that have been discussed in recent studies.

Since DE is one of the most well-known and successful evolution-based metaheuristic algorithms for solving the single-objective bound-constrained problem (SOP), many studies have proposed different ways to enhance its performance, such as an improved version of DE (Tanabe and Fukunaga, 2013, 2014) or combining it with other methods (Lin *et al.*, 2023; Zhang *et al.*, 2025). The transition operator (e.g. mutation) of DE has

a strong impact on its search performance, just like the transition operators of the other search algorithms in the family of metaheuristic algorithms. Consequently, using two or more transition operators in a DE represents a viable approach to improving its search performance. Recently, an effective way to dynamically determine a suitable transition operator has been presented for the DE in solving an optimization problem (Tan *et al.*, 2021, 2022b). The basic idea is to adopt the fitness landscape analysis (FLA) to understand the landscape characteristics of the optimization problem (such as roughness, neutrality, or gradient) first and then use this information to generate a learning model for the metaheuristic algorithm to further select the suitable search operator for different kinds of optimization problems. In other words, the machine learning algorithm will be used to understand the possible search performance of DE mutation operators for the landscape characteristics of the problem, and the learning model generated by the machine learning algorithm will be used to select a suitable mutation operator for different optimization problems. For example, the learning model generated by the machine learning algorithm will be used to determine the mutation operator before the search process begins (Tan *et al.*, 2021). In a later study, Tan *et al.* (2022b) presented a similar method to analyse the landscape characteristics and use a learning model to determine the mutation operator at a fixed number of generations during the convergence process. Of course, using reinforcement learning (RL) as the machine learning component of such an improved DE-based algorithm is also a viable approach (Tan and Li, 2021; Tan *et al.*, 2022a).

Based on our observations, most DEs (Tan *et al.*, 2021; Tan and Li, 2021; Tan *et al.*, 2022a, 2022b) do not account for: (1) the fact that the search state of DE might change after a different number of generations during every state change in the convergence process and (2) the landscape characteristics around the current searched solutions to further select the suitable mutation operator for DE in later generations. Therefore, we believe that there is still plenty of room to enhance the search performance of DE in solving single-objective optimization problems. Motivated by these observations, an effective DE based on FLA will be presented in this study that considers adaptively adjusting the timing to (1) analyse the landscape characteristics around current searched solutions and (2) use this landscape information to further select the appropriate mutation operator during the search process. The main contributions are as follows.

- A new method is proposed to generate detailed landscape characteristics and recommend the suitable mutation operator for the current search state.
- The analysis region is adaptively reduced during the search process to analyse the landscape characteristics in detail, as well as using more random walks for complicated landscapes and fewer random walks for simpler landscapes.
- When the search region differs from the previous analysis region, the proposed algorithm will re-analyse the landscape characteristics to determine the mutation operator, which can be used to reduce the number of times FLA is used in RL-based DE.

The remainder of the paper is organized as follows. Section 2 begins with the problem definition of the single-objective bound-constrained problem (SOP), followed by a brief review of FLA and metaheuristic algorithms based on FLA for solving optimization problems. Section 3 starts with the basic idea of the proposed algorithm, followed by a detailed

description of each operator. To evaluate the performance of the proposed algorithm, we first compare a set of DEs with the proposed algorithm in Section 4. Moreover, we also compare the proposed algorithm with other state-of-the-art FLA-based DEs. Section 5 gives the conclusions of this research and shows some possible research directions of the proposed algorithm.

2. Related Work

2.1. *Problem Definition*

As one of the most well-known optimization problems, the SOP can be regarded as a representative optimization problem in the continuous space. This kind of optimization problem can also be found in many real-world applications. An SOP can generally be defined as follows (Mohamed *et al.*, 2020; Kumar *et al.*, 2021):

$$\min_{s\in\mathbb{A}} f(s), \text{ subject to } s = (s_1, s_2, \ldots, s_d), \text{ and}$$
$$\begin{cases} L_i \leqslant s_i \leqslant U_i, & i \in \{1, 2, \ldots, d\}, \\ h_j^a(s) \leqslant 0, & j \in \{1, 2, \ldots, p\}, \\ h_k^b(s) = 0, & k, \end{cases} \tag{1}$$

where $f()$ is the objective function to be minimized or maximized, $s = (s_1, s_2, \ldots, s_d)$ is a feasible solution in the solution space, $\mathbb{A}$ is the solution space of the optimization problem, and d is the number of dimensions of s. Moreover, L_i and U_i are the lower and upper bounds of the i-th dimension of s, respectively, $h_j^a()$ is the j-th inequality constraint, while $h_k^b()$ is the k-th equality constraint. From this definition, it can be easily seen that the aim of SOP is to find a solution that has the minimal or maximal objective value within the limited space (lower and upper bounds of each dimension) and that satisfies the optional constraints h^a and h^b. The finding process of a metaheuristic algorithm for solving the SOP is usually called the search (or convergence) process. Also, because most SOPs are simple and can be easily realized, many recent studies (Tan *et al.*, 2021, 2022b; Tong *et al.*, 2021; Rani *et al.*, 2024) used them as the touchstone to understand the performance of a metaheuristic algorithm.

2.2. *Fitness Landscape Analysis*

How to describe and display the landscape of the solution space can be dated back to the 1930s or even earlier. Among them is a very early study (Wright, 1932), in which Wright used contour lines to show the fitness values of a set of chromosomes to visualize a two-dimensional fitness landscape metaphor for evolutionary biology. From this perspective, valleys, peaks, ridges, and plateaus can be used to visualize the kind of fitness landscape a search algorithm confronts, as shown in Fig. 1. More precisely, in this example, "+" and "−" represent peaks and valleys, while darker colours represent lower fitness

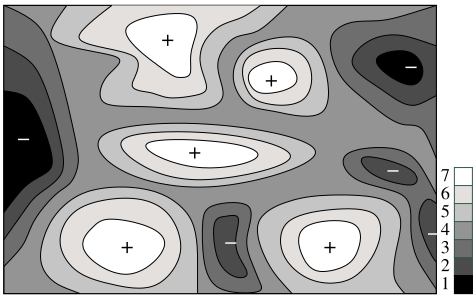
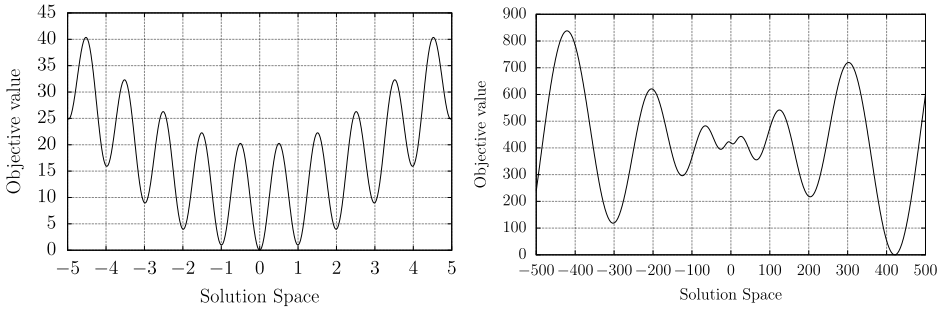


Fig. 1. A simple example of Wright's fitness landscape (Wright, 1932).

values and vice versa. Several later studies (Grefenstette, 1999; Malan and Engelbrecht, 2013) attempted to visualize the relationships between feasible solutions, fitness values of solutions, and the landscape of the solution space to further understand the search performance of a metaheuristic algorithm for various optimization problems. A general and formal definition of the fitness landscape, resting on three basic assumptions, can be found in Stadler (2002):

- a current solution s in the solution space,
- a candidate solution v that is the neighbour of the current solution s in the solution space, and
- a fitness function $f(s)$.

With this definition and the three assumptions, we can easily visualize the landscape of a discrete solution space. For instance, a candidate solution v can be generated from the current solution s by a transition operator in such a way that the Hamming distance between these two solutions is one. This definition can also be used for the continuous landscape. For example, a candidate solution v can also be a neighbour of the current solution s so that the Euclidean distance between these two solutions is less than a predefined small value (Stadler, 2002; Malan and Engelbrecht, 2013). Other studies (Pitzer and Affenzeller, 2012; Ochoa and Malan, 2019; Zou *et al.*, 2022) also focus on understanding the fitness landscape of the solution space of an optimization problem via different perspectives when we want to use a metaheuristic algorithm to solve it. For example, the information from FLA can be used to know the ruggedness (Weinberger, 1990; Malan and Engelbrecht, 2009) or difficulty (Jones and Forrest, 1995) of a search algorithm for solving a particular optimization problem. This means that we can use the characteristics of the solution space of an optimization problem extracted by FLA (e.g. local optima and modality, basins of attraction and funnels, ruggedness and neutrality, gradients and deception, and searchability) to further design a better search algorithm. Nowadays, several methods for characterizing fitness functions and landscapes have been discussed in various studies (Malan and Engelbrecht, 2013; Richter and Engelbrecht, 2014; Ochoa and Malan, 2019) to understand the ruggedness, neutrality, and difficulty of the fitness landscape, such as autocorrelation function (Weinberger, 1990), ruggedness of information entropy (RIE) (Vassilev *et al.*, 2000), neutral walk (Reidys and Stadler, 2001), and fitness distance correlation (FDC) (Jones and Forrest, 1995).



(a) Rastrigin (b) Schwefel 2.26

Fig. 2. The ruggedness of different test functions.

Among them, the autocorrelation function, presented in Weinberger (1990), is an early method based on random walks to understand the "ruggedness" of the binary fitness landscape. Figure 2 shows the landscapes of Rastrigin and Schwefel 2.26 test functions, respectively, which explain that different test functions might have different kinds of ruggedness. The autocorrelation function assumes that the behaviours of a random walk on a landscape will be the same even if they started from different points. Based on this assumption, the autocorrelation function will then use a sequence of fitness values generated by the random walk process on the landscape to calculate the correlation with the same sequence of values a small distance away to further know the ruggedness of the fitness landscape. By using this method, the fitness landscape of a solution space is smooth when the autocorrelation is high and vice versa. Vassilev *et al.* (2000) presented the ruggedness of information entropy (RIE) to measure the ruggedness of the fitness landscape. This RIE metric, of course, also uses a random walk process to sample a set of solutions. A string $\mathbf{s}(\epsilon) = \{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_n\}$ in RIE will be used to represent the ruggedness of the fitness landscape. An element $\mathbf{s}_i$ of $\mathbf{s}(\epsilon)$ can be defined as:

$$\mathbf{s}_i = \begin{cases} \bar{1}, & \text{for } f_i - f_{i-1} < -\epsilon, \\ 0, & \text{for } |f_i - f_{i-1}| \leqslant \epsilon, \\ 1, & \text{for } f_i - f_{i-1} > \epsilon, \end{cases} \tag{2}$$

where ϵ is a real number to determine the accuracy of the calculation of the string $\mathbf{s}(\epsilon)$ which is the observation value of the fitness landscape, n is both the number of solutions and the number of sampling steps of the random walk process, and f_i is the objective value of the i-th sampling solution by a random walk process.

Malan and Engelbrecht (2009) proposed a different approach for measuring the ruggedness of the fitness landscape of an optimization problem, in which strings are generated between predefined maximum and minimum values. The string $\mathbf{s}(\epsilon)$, based on Eq. (2) with ϵ defined by

$$\epsilon = \left\{0, \frac{\epsilon^*}{128}, \frac{\epsilon^*}{64}, \frac{\epsilon^*}{32}, \frac{\epsilon^*}{16}, \frac{\epsilon^*}{8}, \frac{\epsilon^*}{4}, \frac{\epsilon^*}{2}, \epsilon^*\right\} \tag{3}$$

Table 1
Ruggedness of information entropy analysis for different states of the fitness landscapes.

Substring (s_i, s_{i+1})	Landscape Shape	Type
0 0		Neutral
0 1		Rugged
0 $\bar{1}$		Rugged
1 0		Rugged
1 1		Smooth
1 $\bar{1}$		Rugged
$\bar{1}$ 0		Rugged
$\bar{1}$ 1		Rugged
$\bar{1}$ $\bar{1}$		Smooth

and ε^* by

$$\epsilon^* = \max\left(|f_2 - f_1|, |f_3 - f_2|, \ldots, |f_n - f_{n-1}|\right), \tag{4}$$

is used to calculate the information entropy value for generating nine different types of strings to map different states of the fitness landscapes, as shown in Table 1. This means that whether the fitness landscape is rugged, neutral, or smooth can then be measured by three neighbour candidate solutions.

The entropic measure $H(\epsilon)$ is calculated as follows:

$$H(\epsilon) = -\sum_{a \neq b} (P_{[ab]} \log_6 P_{[ab]}), \tag{5}$$

where a and b, elements from the set $\{\bar{1}, 0, 1\}$, are adjacent elements in the string $\mathbf{s}(\epsilon)$, and $P_{[ab]} = n_{[ab]}/n$ with $n_{[ab]}$ representing the number of sub-blocks $[ab]$ appearing in the string $\mathbf{s}(\epsilon)$. In summary, by using Eq. (5) to measure the fitness landscape, a higher value of $H(\epsilon)$ implies a more rugged landscape in the random walk process.

Another type of FLA measurement can be found in Reidys and Stadler (2001), in which it is called the neutral walk and can be used to understand the "neutrality" of the fitness landscape. This method will start with a random solution s_0 in the search space and use the neutral walk process to generate all neutral neighbour solutions around s_0. The fitness values of these neutral neighbour solutions (e.g. s_1) from a neutral walk process need to be the same or similar if and only if $f(s_0) = f(s_1)$. This neutral walk process will continue until it cannot find more neutral neighbours farther away than the current neutral neighbours. The number of steps in the neutral walk will then be used as a measure of the neutrality of the fitness landscape. The fitness distance correlation (Jones and Forrest, 1995) provides an alternative way to measure the "difficulty" of an optimization problem for the search algorithm. The basic idea is to define the correlation coefficient between the objective values of solutions and the distances of the solutions from their corresponding

best-searched solutions as follows:

$$\mathcal{F} = \frac{\text{cov}(F, D)}{\sigma_F \sigma_D}, \tag{6}$$

where $\mathcal{F}$ is the correlation coefficient value, $F = \{f_1, f_2, \ldots, f_n\}$ is a set of objective values of solutions, and $D = \{d_1, d_2, \ldots, d_n\}$ is the corresponding distances of those solutions to the optimal solution. Besides, $\text{cov}(F, D) = \frac{1}{n}\sum_{i=1}^{n}(f_i - \bar{f})(d_i - \bar{d})$ is the covariance between the objective values F and distances D, where σ_F, σ_D, $\bar{f}$, and $\bar{d}$ are the standard deviations and means of F and D, respectively. The value of $\mathcal{F}$ is in the range $[-1, +1]$, with a lower value implying that the analysed problem is an easy maximization problem while a higher value implying that the analysed problem is an easy minimization problem. This means that the optimization problem can be easily solved if the objective (fitness) values of the solutions are similar and their distances are small.

Shen and He (2010) also presented an FLA method for measuring terrain fitness called the "number of optimal values," which is computed from the fitness values of individuals in the population after sorting. Assume that the population is $s = (s_1, s_2, \ldots, s_n)$, where s_i is the i-th solution and n is the number of solutions in s. Further assume that each solution in the population is d-dimensional (i.e. contains d subsolutions); that is, $s_i = (s_{i,1}, s_{i,2}, \ldots, s_{i,d})$, where $s_{i,j}$ represents the j-th dimension of the i-th solution in the population. We can then compute the distances between all solutions and the best one. All these distances will then be sorted in ascending order, yielding the sequence $\{k_1, k_2, \ldots, k_n\}$. This means that each solution s_i is associated with an order index k_j with $i \neq j$, so that the value of k_j is i. Subsequently, the fitness values of individuals are compared in their sorted order. In case that $f_{(k_{i+1})} \leqslant f_{(k_i)}$, the counter χ will be incremented by one. After completing the comparison of all individuals, this count is normalized by the population size as follows:

$$\phi = \frac{\chi}{n}, \tag{7}$$

where ϕ can be used as an approximation of terrain roughness. When $\phi = 0$, the structure is closer to unimodal, which makes the problem relatively easy to solve; conversely, when $\phi = 1$, there are many "good" solutions in the region, resulting in terrain with varying elevations and greater problem complexity.

2.3. *Fitness Landscape Analysis for Metaheuristics*

Many studies (Tsai and Chiang, 2023; Talbi, 2021) found that metaheuristic and machine learning algorithms can complement each other; thus, the demarcation lines between these two different learning algorithms are becoming increasingly obscure and ambiguous. For example, we can use a metaheuristic algorithm to enhance the performance of a machine learning algorithm, such as finding a set of suitable hyperparameters for the deep learning algorithm or generating a suitable deep learning architecture (Tsai and Chiang, 2023). On the contrary, a machine learning algorithm can also be used to improve the performance

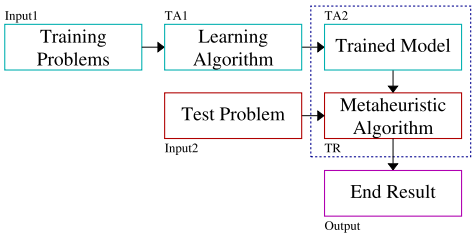


Fig. 3. A simple example of offline learning model for the metaheuristic algorithm.

of a metaheuristic algorithm, such as finding a suitable search operator for the metaheuristic algorithm in different search states or optimization problems. Talbi (2021) divides the ways of using machine learning as a supplementary mechanism for a metaheuristic algorithm into two categories: "offline" and "online," respectively. Most of them imply that we can first use machine learning to roughly know the data distribution, problem characteristics, or even landscape of solution space and suitable search operator for different search states and then use the trained model to help the metaheuristic algorithm automatically determine the appropriate search operator in solving an optimization problem.

2.3.1. *Offline Learning for the Metaheuristic Algorithm*

Using machine learning as an offline mechanism is one of the possible solutions for the metaheuristic algorithms (Merz and Freisleben, 1999; Shen and He, 2010; Wang *et al.*, 2018; Tan *et al.*, 2021; Talbi, 2021; Tan *et al.*, 2022b; da Costa Oliveira *et al.*, 2023; Zhou *et al.*, 2023). Figure 3 provides a simple example to illustrate how a machine learning algorithm is used as an offline mechanism to enhance the performance of a metaheuristic algorithm. In this example, the training process contains the "Input1" as training functions and "TA1" and "TA2" as steps for constructing the trained model, while the test process contains the "Input2" as the unseen functions and "TR" as steps for using the metaheuristic algorithm based on the trained model to determine the suitable search strategy to solve a new optimization problem.

In this way, the learning algorithm will first train the model via the input training problems from TA1 to TA2. Then, the metaheuristic algorithm will use the trained model to determine the suitable search strategy (e.g. search operator) for solving the new test problem and output the final result. Using the trained model of a machine learning algorithm to generate the elite solutions as the initial seeds of the metaheuristic algorithm and to select suitable search operators for designing a good metaheuristic algorithm are two representative research directions of the offline category (Talbi, 2021). Among them, using the information of the fitness landscape of the solution space to enhance the ability of machine learning for improving the end results of a metaheuristic algorithm has been a promising research topic in recent years. In an early study (Merz and Freisleben, 1999), Merz and Freisleben discussed the information of FLA that can be used to design components of a memetic algorithm in solving combinatorial optimization problems, as well as other metaheuristic algorithms. In Shen and He (2010), Shen and He attempted to use

a training model with fitness landscape information to let evolutionary programming understand the relationship between the local fitness landscape and mutation strategies to determine suitable mutation operators for different search states during the convergence process. Wang *et al.* (2018) used the two fitness landscape measurements to measure the evolutionary probability (i.e. the population can evolve to a fitter state) and evolutionary ability (i.e. degree of evolution, fitness improvements, and diversity) of the population to adjust the search strategy during the convergence process.

Tan *et al.* (2021) presented a DE with an adaptive mutation operator based on the information of fitness landscape, called FLDE. The algorithm uses the random forest algorithm to train a learning model to evaluate which mutation operator is suitable for using which fitness landscape characteristic. So FLDE will be based on the fitness landscape characteristic of an optimization problem to further determine a suitable mutation strategy before the search process begins. In a later study, Tan *et al.* (2022b) further used the k-nearest neighbours (kNN) algorithm for constructing the relationship between dynamic fitness landscape characteristics (i.e. distance for each individual to the optimal global solution and information entropy to measure ruggedness of solution space) and mutation operators for determining the suitable mutation operators during the search process. Zhou *et al.* (2023) also attempted to use the FLA information to enhance the searchability of the artificial bee colony called FLABC. In this algorithm, the population can be regarded as a sample of the fitness landscape while the dispersion metric (i.e. average pairwise Euclidean distance between the best searched solutions) will be used to identify the landscape characteristics. Based on the result of the dispersion metric, FLABC can then know whether the local landscape is smooth or rugged. If the observation of the local landscape is smooth, the search behaviour of FLABC will then move toward "exploitation." In case the observation of the local landscape is rugged, the search behaviour will tend to "exploration."

2.3.2. *Online Learning for the Metaheuristic Algorithm*

In addition to using machine learning as an offline mechanism to enhance metaheuristic algorithm performance, it can also serve as an online mechanism to provide supplemental information for a metaheuristic algorithm to select a suitable transition operator and update the model incrementally during the convergence process, which is an alternative approach for metaheuristic algorithms (Zhang *et al.*, 2023; Seyyedabbasi, 2023; Tessari and Iacca, 2022; Tan *et al.*, 2022a). Figure 4 provides a simple example to illustrate how a machine learning algorithm is used as an online mechanism to enhance the performance of a metaheuristic algorithm. In this example, the "Input" is the unseen functions, and the TA step represents a learning model. When the metaheuristic algorithm is used to solve a new optimization problem, it will use the learning model to determine the suitable search strategy (i.e. the TR step). Then, the search results will be used to update the learning model and make the recommendation of the learning model more accurate in later iterations. In this example, of course, the learning model can also be trained by using the given training problems before it is used for the metaheuristic algorithm; however, the main focus is on incrementally updating the learning model with the current searched results from the metaheuristic algorithm.

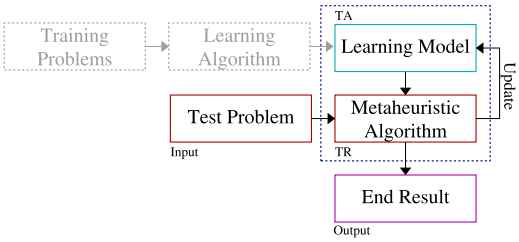


Fig. 4. A simple example of online learning model for the metaheuristic algorithm.

In this online category, reinforcement learning (RL) is a representative learning algorithm (Zhang *et al.*, 2023; Seyyedabbasi, 2023; Tessari and Iacca, 2022) because the RL model can be used to help the metaheuristic algorithm determine the appropriate search strategy (e.g. search operator) in different periods of the convergence process. This means that the RL model will be learned and updated based on the new searched results during the convergence process. In Seyyedabbasi (2023), Seyyedabbasi utilized a Q-learning algorithm (a variant of the reinforcement learning algorithm) to help sand cat swarm optimization algorithm (SCSO) tend to the "exploration" or "exploitation" on different search states of the convergence process. Tessari and Iacca (2022) presented a general-purpose framework by using the reinforcement learning algorithm for performing parameter adaptation in metaheuristic algorithms, such as step-size of covariance matrix adaptation evolution strategies (CMA-ES) and crossover mutation of differential evolution (DE).

2.4. *Summary*

The above discussion demonstrated that FLA methods provide several means to characterize optimization problems, such as whether the landscape is smooth or rugged. With the fitness landscape information, we can then provide a metaheuristic algorithm some additional "vision" to better understand the landscape of the solution space. Therefore, it might be able to enhance the searchability of the metaheuristic algorithm, and that is why some recent studies (Merz and Freisleben, 1999; Shen and He, 2010; Wang *et al.*, 2018; Tan *et al.*, 2021, 2022b; Zhou *et al.*, 2023) attempted to combine the FLA with the metaheuristic algorithm for solving optimization problems. However, even though measurements of FLA might be able to provide us some possible ways to better understand the landscape of the solution space, they still have limitations because no FLA method can exactly depict the whole solution space. This means that most FLA methods can only roughly depict the whole/local solution space, but it can still provide some additional fitness landscape information to enhance the search performance of the metaheuristic algorithm compared with the metaheuristic algorithm without any fitness landscape information. That is why some studies (Wang *et al.*, 2018; Tan *et al.*, 2021) attempted to use two or more FLA methods to understand the solution space.

In addition to the accuracy of FLA, how to use this information to select the suitable search algorithm or operator for the metaheuristic algorithm is another critical research issue that has attracted the attention of researchers in recent years. This research topic can

be regarded as knowing which algorithm or operator will suit the detected local fitness landscape. Of course, we can expect that a good way to map the search operator with the fitness landscape will enhance the metaheuristic algorithm performance if we have many search operators and many kinds of local fitness landscapes of the solution space. Among them, most combinations of metaheuristic algorithms with FLA determine the search strategy (or operator) before the search process begins (Tan *et al.*, 2021), using the whole solution space information to further adjust the search strategy (Tan *et al.*, 2022b) or using a predefined rule to determine the search strategy, as in Zhou *et al.* (2023). That is why this research focuses on developing an effective approach to use a machine learning algorithm based on the FLA information of partial solution space for the metaheuristic algorithm to adaptively determine the suitable search operator for different search states during the convergence process.

3. Proposed Algorithm

3.1. *Basic Idea*

This paper introduces an effective differential evolution algorithm with an adaptive mutation operator based on the information from the reinforcement learning model and FLA, called reinforcement learning based differential evolution with fitness landscape analysis (RLDEFLA). It is inspired by the studies of Tan *et al.* (2022b), Tan and Li (2021) because both use FLA to improve the performance of metaheuristic algorithms. In the studies of Tan *et al.* (2022b), Tan and Li (2021), an ML or RL algorithm is used to construct the relationship between landscape characteristics and mutation operators in the offline phase (training process). The trained model generated by the offline phase (training process) is then used to determine suitable mutation operators for DE before or during the convergence process. However, according to our observations, both studies (Tan *et al.*, 2022b; Tan and Li, 2021) only use the random walk procedure to understand the fitness landscape of the whole solution space; the designs are not for the partial solution space around the current searched solution. We believe that the search strategy adjustment of DE during the convergence process needs to be based on the fitness landscape information around the current search solution to make such adjustments more accurate in solving an optimization problem. It is well known that the fitness landscape of the solution space might contain a couple of different landscapes in different regions. Hence, the basic idea of the proposed algorithm is to adjust the search strategy when the metaheuristic algorithm confronts different fitness landscape states (i.e. search state) in the "particular region of the solution space" instead of the "whole solution space." This means that the search state information will be used to determine whether the landscape needs to re-analyse the landscape characteristic and change the search operator. Furthermore, the proposed algorithm will dynamically adjust the number of steps in the next random walk based on the FLA results obtained from the previous random walk, either to enable more random walks for comprehensive exploration when the landscape is complicated or to reduce the number of random walks when the landscape is simpler.

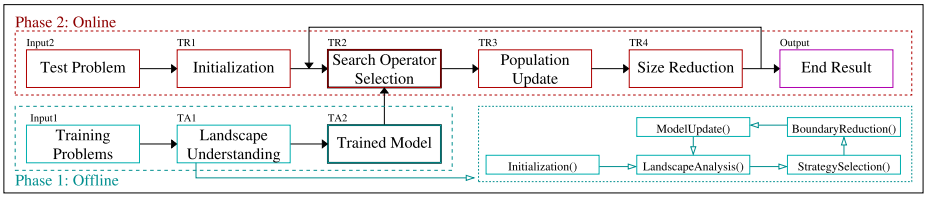


Fig. 5. The flow chart of RLDEFLA.

Figure 5 shows the system design of the proposed algorithm, RLDEFLA. It can be divided into two phases: offline and online. The first phase is the offline and can be regarded as a "supervised learning" process, whereas the proposed algorithm employs reinforcement learning during training. In this phase, RLDEFLA will use the random walk procedure from a set of training functions (i.e. Input 1) to understand which search strategy will be useful for which fitness landscape (i.e. TA1). More precisely, TA1 represents the landscape understanding procedure which contains the following modules: Initialization(), LandscapeAnalysis(), StrategySelection(), BoundaryReduction(), and ModelUpdate(). Subsequently, in TA2, the proposed algorithm will train a deep Q-learning (DQN) model with the results from TA1 via reinforcement learning. The goal of this step is to learn a better strategy for selecting search strategies under different landscape conditions.

The second phase is the online phase which can be regarded as an "unsupervised search" process. The new test function (i.e. Input 2) and learning model from phase 1 will be used as the inputs to the metaheuristic algorithm of the online phase. Note that the DE will be used as the metaheuristic algorithm in this study. Both inputs (i.e. Input 2 and learning model of TA2) will be used to select a suitable search operator (e.g. mutation operator of the DE) in the TR2. Moreover, the proposed algorithm then uses this mutation operator to update the population and uses a size reduction mechanism to enhance search performance during TR3 and TR4. At the end of the convergence process of this phase, the proposed algorithm will output a good result for the input test problem.

3.2. *Phase 1: Offline Supervised Learning*

Algorithm 1 shows the offline supervised learning phase of the proposed algorithm that contains the following operators: Initialization(), LandscapeAnalysis(), StrategySelection(), BoundaryReduction(), and ModelUpdate().

In addition to inputting the information of training problems to the Initialization() operator, a set of different operators is also randomly generated for the solution space by this operator. The LandscapeAnalysis() operator analyses the landscape characteristics based on the current search region. The StrategySelection() operator will use a learning model (e.g. DQN model) to select a suitable search operator, such as a mutation operator. The BoundaryReduction() operator is tasked to narrow down the analysis region based on the search state. The ModelUpdate() operator will then update the weights of the learning model using the reward from recent search results. All operators in the construction method of the landscape search strategy are described in detail below.

Algorithm 1: Offline Supervised Learning

```
Initialization()
while the termination condition is not met do
    LandscapeAnalysis()
    StrategySelection()
    BoundaryReduction()
    ModelUpdate()
end
```

3.2.1. *Initialization()*
This operator randomly generates a set of solutions $s = (s_1, s_2, \ldots, s_n)$ in the solution space, where n is the number of individuals in the population. Then, the objective value of each individual is calculated using the objective function. The analysis upper bound U^a and analysis lower bound L^a are set equal to the upper bound U and lower bound L of the input training problems,[1] respectively.

3.2.2. *LandscapeAnalysis()*
The main goal of this operator is to analyse the landscape characteristics of input training problems. It employs the random walk algorithm to sample the solution space to create a random walk path. The number of steps n_s and the step size α need to be specified before using the random walk algorithm. The step size is defined as:

$$\alpha_j = \frac{(U_j^a - L_j^a)}{\bar{\alpha}}, \tag{8}$$

where α_j is the step size of the j-th dimension; $\bar{\alpha}$ is set equal to 10.0 according to Tan *et al.* (2021); U_j^a and L_j^a denote, respectively, the analysis upper bound and the analysis lower bound. A random walk path $P = \{p_1, p_2, \ldots, p_{n_s}\}$ is generated as follows:

$$p_{i+1,j} = p_{i,j} + \text{rand}(0, 1) \cdot \alpha_j, \tag{9}$$

where $p_{i,j}$ and $p_{i+1,j}$ are the values of the j-th dimension of the solution in the random walk path. If a solution is not in the analysis range during the generation of a random walk path, it will be adjusted as follows:

$$p_{i+1,j} = \begin{cases} p_{i+1,j} - (U_j^a - L_j^a), & \text{if } p_{i+1,j} > U_j^a, \\ p_{i+1,j}, & \text{otherwise.} \end{cases} \tag{10}$$

After generating the random walk path, the landscape characteristics are calculated. The encoding of the landscape characteristic vector v is defined as:

$$v = \langle g, \tau, \phi, \rho, R_f \rangle, \tag{11}$$

[1] In this study, the input training problems can be regarded as the training dataset of a machine learning algorithm.

where g represents the current generation, τ represents the autocorrelation value, ϕ is the number of optimal values, ρ is the fitness distance correlation value (Jones and Forrest, 1995), and R_f denotes the maximum ruggedness of information entropy value (Malan and Engelbrecht, 2009). The landscape characteristics for each random walk path are computed, with each element in the landscape characteristic vector ν representing the average value of the corresponding metrics. These generated landscape characteristics serve as features for the dataset.

3.2.3. *StrategySelection()*

In this study, one of the three mutation strategies will be selected using the DQN model, as defined in Eq. (12). The mutation strategy V^* will be randomly selected from the three options if the random value φ is less than the predefined threshold t_h; otherwise, the mutation strategy will be selected by using DQN with the highest weight in its model.

$$V^* = \begin{cases} V^{\text{rand}}, & \text{if } \varphi < t_h, \\ \arg\max\limits_{\mathcal{V}} Q(s, a), & \text{otherwise}, \end{cases} \tag{12}$$

where φ is a random value in [0, 1], t_h is a predefined threshold (it is set equal to 0.05 in this study), and $\mathcal{V} \in \{V^1, V^2, V^3\}$. The three different mutation operators (i.e. DE/rand/1, DE/current-to-rand/1, and DE/current-to-pbest/1) are defined as

$$V^1_{i,j} = X_{r_1,j} + f_i \cdot (X_{r_2,j} - X_{r_3,j}), \tag{13}$$

$$V^2_{i,j} = X_{i,j} + f_i \cdot (X_{r_1,j} - X_{i,j}) + f_i \cdot (X_{r_2,j} - X_{r_3,j}), \tag{14}$$

$$V^3_{i,j} = X_{i,j} + f_i \cdot (X_{pbest,j} - X_{i,j}) + f_i \cdot (X_{r_1,j} - X_{r_2,j}), \tag{15}$$

where $X_{i,j}$ is the j-th dimension of the i-th individual; X_{pbest} is randomly chosen from the top $p\%$ of individuals; $V_{i,j}$ is the mutant vector of $X_{i,j}$; $X_{r_1,j}$, $X_{r_2,j}$, and $X_{r_3,j}$ where $r_1 \neq r_2 \neq r_3$ are individuals randomly selected from the population. These three different mutation operators will search for a fixed number of generations I. Each mutation operator can obtain the R search results after being carried out for R runs.

3.2.4. *BoundaryReduction()*

To imitate the converging process that the search region will tend to a smaller region gradually, this operator will reduce the analysis region to analyse the landscape characteristics of the particular region in detail. This operator updates the minimum and maximum values of each dimension in the entire population for the current analysis region. The maximum and minimum values will be, respectively, the analysis upper and lower bounds of the corresponding dimension defined as follows:

$$L^a_j = \min X^g_{i,j}, \tag{16}$$

$$U^a_j = \max X^g_{i,j}, \tag{17}$$

where L_j^a and U_j^a represent the analysed results of L_j and U_j, respectively, and they will be used to control the lower and upper bounds of the j-th dimension of searches of the proposed algorithm, and $X_{i,j}^g$ represents the j-th dimension of the i-th individual at the g-th generation.

3.2.5. *ModelUpdate()*

The design of this operator is used to update the DQN weights. It means that after each DE generation, the evolutionary efficiency e_p is computed from the fitness of new solutions, which will be used as a reward in reinforcement learning to update the DQN model. In this operator, χ_i is a survival counter for the i-th individual, as shown in Eq. (18) (Tan *et al.*, 2022a). If the fitness value of a trial vector (a new individual) $f(V_i)$ is better than the fitness value of its parent vector[2], its survival count χ_i will be incremented by one; otherwise, it is reset to zero.

$$\chi_i = \begin{cases} \chi_i + 1, & \text{if } f(V_i) \leqslant f(X_i), \\ 0, & \text{otherwise.} \end{cases} \tag{18}$$

As shown in Eq. (19), e_p denotes the evolutionary efficiency of the population, defined as the ratio of the total survival count of all individuals to the current population size n.

$$e_p = \frac{\sum_{i=1}^{n} \chi_i}{n}. \tag{19}$$

Then, the reward is $R_{t+1} = e_p$. With such information, we can then compute the reward to update the DQN model, as follows:

$$Q(S_t, A_t) \leftarrow (1-\alpha)Q(S_t, A_t) + \alpha\Big(R_{t+1} + \gamma \max_a Q(S_{t+1}, a)\Big), \tag{20}$$

where $Q(S_t, A_t)$ represents the behavioural value for the action A_t in state S_t, α is the learning rate, γ is the discount factor to measure the importance of future rewards, the maximum value of $Q(S_{t+1}, a)$ represents the maximum expected value corresponding to all possible actions in the next state S_{t+1}.

3.3. *Phase 2: Online Unsupervised Search*

After training the machine learning model of phase 1, the proposed algorithm will use it to recommend the mutation operator of DE during the search process of phase 2. As shown in Algorithm 2, the search process of online unsupervised search algorithm starts with the Initialization() operator and contains three operators: StrategySelection(), PopulationUpdate(), and SizeReduction().

[2]Note that the smaller fitness value stands for better quality of a solution in this study.

Algorithm 2: Online Unsupervised Search

```
Initialization()
while the termination condition is not met do
    StrategySelection()
    PopulationUpdate()
    SizeReduction()
end
```

The Initialization() operator randomly generates a population of solutions in the solution space. The StrategySelection() operator is responsible for determining the recommended mutation operator. The trained model is used in this operator. The PopulationUpdate() operator is used to update the population using the recommended mutation operator, which is determined by the StrategySelection() operator. The SizeReduction() operator takes care of reducing the population size and improving the local search ability. Note that Algorithm 2 is built upon LSHADE, but instead of its original mutation operator, we employ three different mutation operators, as shown in Eqs. (13), (14), and (15), and remove the external archive. We refer to this new algorithm as the modified LSHADE.

3.3.1. *Initialization()*

In the initialization operator, the population $X = \{X_1, X_2, \ldots, X_n\}$ is randomly generated in the solution space, as follows:

$$X_{i,j} = L_j + \text{rand}(0, 1) \cdot (U_j - L_j), \tag{21}$$

where $X_{i,j}$ is the j-th dimension of the i-th individual; L_j and U_j are the lower and upper bounds, respectively. Moreover, each mutation operator maintains its historical table H to generate the control parameters. All the entries in the historical tables are initialized to 0.5 based on Tanabe and Fukunaga (2013).

3.3.2. *StrategySelection()*

This operator identifies recommended mutation operators for updating the population. To maintain the diversity of the algorithm, the proposed algorithm randomly selects a mutation operator from three mutation operators once after a fixed number of generations. When the current generation g equals a fixed number of generations, the trained model recommends a mutation operator. For analysing the landscape precisely, the analysis upper bound U^a and the analysis lower bound L^a are defined by the maximum and minimum values of the population. The random walk algorithm then generates a random walk path in the analysis range. Different from the original random walk algorithm, the starting point of the random walk path is set to the best solution so far, and the random walk step is calculated as follows:

$$\alpha_j = \frac{(U_j^a - L_j^a)}{\gamma_c}, \tag{22}$$

where α_j is the j-th dimension of the step size, U_j^a and L_j^a are, respectively, the analysis upper and lower bounds, and γ_c is a control value. Note that for Eqs. (21) and (22), L_j and U_j are the lower and upper bounds of the j-th dimension of the solution space of an optimization problem that are inputs by the problem descriptions, and both of them are fixed, while L_j^a and U_j^a represent the analysed results for L_j and U_j that are used to control the search scope of the proposed algorithm during the convergence process. Note also that in this study, the number of evaluations due to all random-walk sampling is counted and included in the overall function evaluation budget (MaxFES) of the proposed algorithm to ensure a fair comparison with other search algorithms. Landscape characteristics, calculated using a random walk path, are composed into a vector, as shown in Eq. (11), to be used as the input for the model. After inputting the landscape characteristic vector to the trained model, it will output the recommended mutation operator to update the solution. The best solution V^* of random walk path p' will be the criterion to decide whether the landscape characteristics change, as shown in Eq. (23). If the worst individual X_w in the population is better than p', the landscape needs to be analysed again. The analysis result will be input into the model to recommend the mutation operator.

$$V^* = \arg\max_{\mathcal{V}} \ Q(s, a), \tag{23}$$

where $\mathcal{V} \in \{V^1, V^2, V^3\}$ represents a set of mutation strategies and V^* has the maximum weights of the DQN. With this model, the proposed algorithm will dynamically adjust the number of steps in the random walk based on the analysis results from the previous FLA. In other words, it will invest more random walk steps in the complex solution space to achieve more comprehensive exploration, and fewer steps in simpler solution-space landscapes (e.g. a flat landscape) to reduce computational cost.

3.3.3. *PopulationUpdate()*

This operator uses the recommended mutation operator to generate the mutant vector V and update the population. The historical memory (table) H generates the control parameters and can be defined as follows:

$$f_i = \text{randc}_i\left(H_{s_r,r_i}^f, 0.1\right), \tag{24}$$

$$c_i = \text{randn}_i\left(H_{s_r,r_i}^c, 0.1\right), \tag{25}$$

where f_i is the scaling factor randomly generated from the Cauchy distribution with mean $H_{s_r}^f$ and variance 0.1; c_i is the crossover rate randomly generated from the normal distribution with mean $H_{s_r}^c$ and variance 0.1; r_i is a random number to randomly pick a value from the historical memory H for the random functions randc_i and randn_i to obtain the scaling factor and crossover rate for the i-th solution, and s_r is a number that represents the recommended mutation operator. When the values of f_i and c_i are not in the range [0, 1], they have to be adjusted (Tanabe and Fukunaga, 2013). After generating the control parameters, the recommended mutation operator is used to generate the mutant vector. When

the generated mutant vector is not in the solution space, it will be adjusted as follows:

$$V_{i,j} = \begin{cases} (X^g_{i,j} + L_j)/2, & \text{if } V_{i,j} < L_j, \\ (X^g_{i,j} + U_j)/2, & \text{if } V_{i,j} > U_j. \end{cases} \tag{26}$$

After that, the crossover and selection operators are used to update the population. During the selection process, the control parameters f_i and c_i are recorded into the success parameter table as S^f and S^c if the trial vector is better than the corresponding individual. The success parameter table is then used to update the historical table as follows:

$$w_i = \frac{\Delta I_i}{\sum_{i=1}^{|S|} \Delta I_i}, \tag{27}$$

$$H^f_{h_{s_r}} = \frac{\sum_{i=1}^{|S^f|} w_i \cdot (S^f_i)^2}{\sum_{i=1}^{|S^f|} w_i \cdot S^f_i}, \tag{28}$$

$$H^c_{h_{s_r}} = \frac{\sum_{i=1}^{|S^c|} w_i \cdot (S^c_i)^2}{\sum_{i=1}^{|S^c|} w_i \cdot S^c_i}, \tag{29}$$

where ΔI_i is the improvement of the objective value of i-th individual; w_i, the weight of S^f_i and S^c_i, is calculated by the improvement of the objective value; h_{s_r} is the historical table counter of the s_r-th mutation operator. Moreover, in this paper, Eqs. (24) and (25) are used to generate the scaling factor f_i and crossover rate c_i; so, both s_r and r_i are needed to represent the associated mutation operator and a random index to the historical memory H. Different from Eqs. (24) and (25), Eqs. (28) and (29) are used to update the historical memory; thus, they only need the index s_r for the particular mutation operator, but not r_i to randomly pick a value from H. The historical tables $H^f_{h_{s_r}}$ and $H^c_{h_{s_r}}$ are updated by calculating the weighted Lehmer mean value. Before the size reduction operator is invoked, the worst individual X_w is recorded and used in the StrategySelection() operator.

To prevent the proposed algorithm from stagnating at a local optimum for an extended period during the search, we design a population restart mechanism. This mechanism monitors the search status by counting the number of consecutive iterations in which no better solution is found, denoted as the number of stagnant generations G_s. When G_s exceeds a predefined threshold G_r, the proposed algorithm replaces the κ worst-performing individuals in the current population with new ones generated by the same Initialization() method described earlier and resets G_s to 0, thereby enabling the population to explore new search directions and enhancing its subsequent exploration capability. The value of κ is determined by the following equation.

$$\kappa = \max\big(1, \lfloor R_r \cdot |X| \rfloor\big), \tag{30}$$

where R_r is the replacement ratio for the population.

3.3.4. *SizeReduction()*

The size reduction operator used in the proposed algorithm is based on the LSHADE (Tanabe and Fukunaga, 2014). In this operator, the population size is reduced throughout the search process to enhance the local search ability. The population size of the size reduction operator is defined as follows:

$$|X| = \text{round}\left(\left(|X_{\min}| - |X_{\text{init}}|\right) \cdot \frac{E}{E_{\max}} + |X_{\text{init}}|\right), \tag{31}$$

where $|X|$ is the population size after reduction; $|X_{\min}|$ is the minimum size of the population; $|X_{\text{init}}|$ is the initial population size; E is the number of evaluations; $E_{\max}$ is the maximum number of evaluations. Size reduction will delete the worst individual in the population until the size of the population is equal to $|X|$.

4. Experimental Results

4.1. *Experimental Environment and Parameter Settings*

All experiments were conducted on a PC with an Intel(R) i9-13900K processor (3.0 GHz, 32 MB cache, and 24 cores), 32GB of memory, and Mesa Intel(R) Graphics (RPL-S) running Linux Ubuntu 22.04.4 LTS. All the programs are written in C++ and compiled using g++ 11.4.0. The machine learning model is written in Python 3.10.12. The proposed algorithm is compared with seven algorithms: the adaptive differential evolution with optional external archive (JADE) (Zhang and Sanderson, 2009), SHADE (Tanabe and Fukunaga, 2013), LSHADE (Tanabe and Fukunaga, 2014), FLDE (Tan *et al.*, 2021), DEDQN (Tan and Li, 2021), RLHPSDE (Tan *et al.*, 2022a), and DFLDE (Tan *et al.*, 2022b) on the CEC2021 and CEC2022 benchmark functions (Mohamed *et al.*, 2020; Kumar *et al.*, 2021). The DQN model used in the proposed algorithm is based on the deep Q-Network (Tan and Li, 2021; Mnih *et al.*, 2015) with 2 fully connected hidden layers, 10 neurons per hidden layer, ReLU activation, 3 neurons for the output layer (each of which is a mutation strategy, i.e. DE/rand/1, DE/current-to-rand/1, and DE/current-to-pbest/1), Adam optimizer, 1×10^{-4} learning rate, mean squared error as the loss function, and mini-batch size of 64. The CEC 2013 benchmark (Liang *et al.*, 2013) is used as a function set to generate training samples as in Tan *et al.* (2021) and Tan *et al.* (2022b). All algorithms are carried out for 51 independent runs. The maximum numbers of evaluations $E_{\max}$ for 10 and 20 dimensions are 200 000 and 1 000 000 according to the settings of Mohamed *et al.* (2020) and Kumar *et al.* (2021). Note that CEC2021 contains 10 basic functions (categorized by their characteristics as unimodal, basic, hybrid, and composition functions), but they have been transformed via "shift," "bias and shift," "shift and rotation," "bias, shift, and rotation" to generate another 40 functions. So there are a total of 50 test functions in CEC2021. Also note that CEC2022 contains 12 test functions.

The parameters in the construction method of the landscape search strategy are set as follows: the number of runs R is set equal to 30; the maximum number of evaluations

Table 2
Parameter settings.

Algorithm	Parameter setting
JADE	$X_{\text{init}} = 100$, $r_{\text{arc}} = 1$, $p = 0.05$, $c = 0.1$
SHADE	$X_{\text{init}} = 100$, $r_{\text{arc}} = 1$, $H = 100$
LSHADE	$X_{\text{init}} = 18\times d$, $X_{\min} = 4$, $r_{\text{arc}} = 2.6$, $H = 6$, $p = 0.11$
FLDE	$X_{\text{init}} = 18\times d$, $X_{\min} = 4$, $r_{\text{arc}} = 2.6$, $H = 5$, $n_s = 2{,}000$, $\alpha = 10$
DFLDE	$X_{\text{init}} = 18\times d$, $X_{\min} = 4$, $I = 100$, $H = 5$, $n_s = 1{,}000$, $\alpha = 20$, $p = 0.1$, $k = 13$
DEDQN	$X_{\text{init}} = 20 \times d$, $X_{\min} = 7$, $I = 1$, $H = 8$, $n_s = 200$, $\alpha = 10$, $p = 0.14$
RLHPSDE	$X_{\text{init}} = 18 \times d$, $X_{\min} = 4$, $I = 1$, $H = 5$, $n_s = 200$, $\alpha = 10$
RLDEFLA	$X_{\text{init}} = 22 \times d$, $X_{\min} = 10$, $I = 110$, $H = 15$, $R_n = 3.1$, $\gamma_c = E_{\max}/100000$, $p = 0.15$, $G_r = 70$, $R_r = 0.05$

Table 3
Search range of hyperparameter optimization for RLDEFLA.

Parameters	Range
X_{init}	$\{15, 16, \ldots, 24, 25\} \cdot d$
$X_{\min}$	$\{5, 6, \ldots, 14, 15\}$
H	$\{5, 6, \ldots, 14, 15\}$
p	$\{0.11, 0.12, \ldots, 0.19, 0.20\}$
I	$\{50, 60, \ldots, 140, 150\}$
R_n	$\{1.3, 1.6, \ldots, 3.7, 4.0\}$
G_r	$\{40, 50, \ldots, 70, 80\}$
R_r	$\{0.05, 0.1, \ldots, 0.2, 0.25\}$

$E_{\max}$ is set equal to $100 \times d$, where d is the number of dimensions; and the step number n_s is set equal to 1000. The parameter settings for all algorithms are shown in Table 2 where X_{init} represents the initial population size, $X_{\min}$ the minimum population size, r_{arc} the rate of the external archive, p the rate of the top individual, c the control parameter to adjust the crossover and mutation rates, H the size of the historical table, n_s the number of steps, α the step size, I a fixed number of generations as an interval to reanalyse the fitness landscape and determine the suitable search strategy, R_n the ratio of the step size, $n_s = E_{\max}/2000 \times R_n$, G_r the restart threshold, and R_r the replacement ratio for the population. Because the number of combinations of hyperparameters X_{init}, $X_{\min}$, H, p, I, R_n, G_r, and R_r of RLDEFLA is excessively large, they are set by the search results of the tree-structured Parzen estimator (TPE) (Bergstra *et al.*, 2011). The search range of these eight parameters is as given in Table 3.

Using TPE for hyperparameter optimization typically requires a measurement to evaluate the quality of a set of parameter settings. In this study, the initial parameter settings for TPE are set to the midpoint values of the predefined search ranges for each parameter. The measurement of the improvement of a set of new parameter settings compared with

Table 4
The initial and best hyperparameter settings of RLDEFLA.

	X_{init}	$X_{\min}$	I	H	p	R_n	G_r	R_r
Initial settings	$20 \cdot d$	10	100	10	0.15	2.5	60	0.15
Settings found by TPE	$22 \cdot d$	10	110	15	0.15	3.1	70	0.05

initial parameter settings is defined as follows:

$$I = \frac{P_o - P_t}{(P_o + P_t)/2} \times 100\%, \tag{32}$$

where P_o and P_t represent the average objective values by using initial parameter settings and parameter settings found by TPE for RLDEFLA, respectively.

Moreover, in this process, the TPE will be used for 500 different sets of hyperparameter settings. The initial hyperparameter settings and best hyperparameter settings found by TPE are shown in Table 4. To better understand the impact of each hyperparameter on the proposed algorithm, we further conduct single-parameter variation experiments for these eight parameters. We first vary one parameter while keeping the remaining parameters fixed, as determined by TPE. Figure 6 uses 0% as the baseline to represent the performance of the parameter combination determined by TPE; the other points show performance relative to this reference setting. These results show that the parameter values determined by TPE yield suitable performance in most cases, and that further improvement may be possible by varying a single parameter over a series of additional tests.

4.2. *Ablation Study of Search Operators*

This section presents the ablation study of the proposed algorithm based on the CEC2021 and CEC2022 benchmark functions. The ablation study focuses on the following mechanisms: timing to FLA (tFLA), step size (SS), and population restart (PR). As shown in Table 5, "Difference" represents the comparison difference between (1) the proposed algorithm using tFLA, SS, and PR mechanisms and (2) the proposed algorithm using some of these mechanisms, in terms of the objective value. Also, "✓" and "–" in this table represent whether the operator is used in the proposed algorithm or not, respectively. The ablation studies show the importance of the mechanisms. It can be easily seen that each of these mechanisms will affect the end result of the proposed algorithm regardless of whether they degrade the end result from 7.25% to 34.57% compared with using all of them in the proposed algorithm. The results further explain why we use these mechanisms in the proposed algorithm.

4.3. *Comparisons with Other Algorithms*

In this section, the CEC2021 and CEC2022 benchmarks, with 10D and 20D, are also used to test the performance of the proposed algorithm and other state-of-the-art search algorithms.

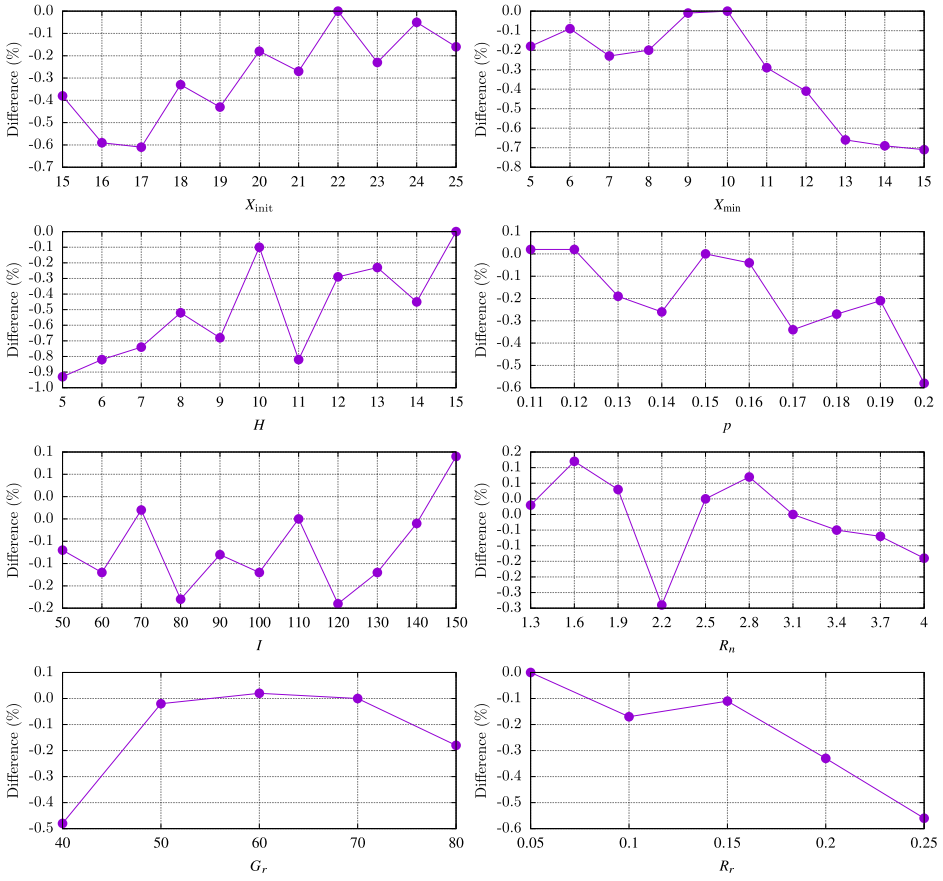


Fig. 6. Parameter sensitivity analysis.

Table 5
Ablation study for operators.

tFLA	SS	PR	Difference
✓	✓	✓	–
✓	✓	–	−12.54%
✓	–	✓	−8.41%
✓	–	–	−14.17%
–	✓	✓	−7.25%
–	✓	–	−17.89%
–	–	✓	−23.43%
–	–	–	−34.57%

4.3.1. *Evaluation Criteria for CEC2021*

The evaluation criterion CEC2021 (Mohamed *et al.*, 2020) is based on the error value and rank of a search algorithm and is defined by

$$S_{21} = S_1 + S_2, \tag{33}$$

$$S_1 = \left(1 - \frac{\text{SNE} - \text{SNE}_{\min}}{\text{SNE}}\right) \times 50, \tag{34}$$

where SNE denotes the average of all normalized error values over all functions and is defined as

$$\text{SNE} = 0.5 \sum_{m=1}^{5} \sum_{i=1}^{10} \text{NE}_{i,m}^{10d} + 0.5 \sum_{m=1}^{5} \sum_{i=1}^{10} \text{NE}_{i,m}^{20d}, \tag{35}$$

$$\text{NE} = \frac{f(s_e) - f(s_b)}{f(s_e)_{\max} - f(s_b)}, \tag{36}$$

where i and m represent the indices of the i-th function and the m-th transformation of a test function, $10d$ and $20d$ are the indices of the 10 and 20 dimensional test functions, respectively, $f(s_e)$ is the best result of a search algorithm out of 51 runs, $f(s_b)$ is the known optimal value of a function, and $f(s_e)_{\max}$ is the largest $f(s_e)$ among all algorithms.

$$S_2 = \left(1 - \frac{\text{SR} - \text{SR}_{\min}}{\text{SR}}\right) \times 50, \tag{37}$$

$$\text{SR} = 0.5 \sum_{m=1}^{5} \sum_{i=1}^{10} \mathbf{r}_{i,m}^{10d} + 0.5 \sum_{m=1}^{5} \sum_{i=1}^{10} \mathbf{r}_{i,m}^{20d}, \tag{38}$$

where $\mathbf{r}$ is the rank of an algorithm among all algorithms for a given test function.

4.3.2. *Evaluation Criteria for CEC2022*

The evaluation criterion for CEC2022 (Kumar *et al.*, 2021) first considers accuracy and then MaxFES to determine the rank of an algorithm for all runs, which can be defined by

$$S_{22} = S_{10} + S_{20}, \tag{39}$$

where S_{10} and S_{20} represent the sum of ranks for all their trials SR for benchmarks of 10D and 20D, respectively. In case two algorithms cannot reach the minimum error value (10^{-8}) before MaxFES is reached, the rank of an algorithm $\mathcal{A}_1$ will be better than that of algorithm $\mathcal{A}_2$ when the error value of $\mathcal{A}_1$ is smaller than that of $\mathcal{A}_2$. In case both algorithms can reach the minimum error value before MaxFES is reached, and the convergence of an algorithm $\mathcal{A}_1$ is faster than another algorithm $\mathcal{A}_2$, then the rank of $\mathcal{A}_1$ will be better than that of $\mathcal{A}_2$. Moreover, all the algorithms will use this approach to get the sum of ranks for all their trials, which is defined as follows

$$\text{SR} = \sum_{i=1}^{R} \mathbf{r}_i, \tag{40}$$

where R is the number of runs. The SR can then be used to compare the performance of the search algorithms.

Table 6
The comparisons between the proposed algorithm and other algorithms for CEC2021.

	JADE	SHADE	LSHADE	FLDE	DFLDE	DEDQN	RLHPSDE	RLDEFLA
S_1	32.89	**50.00**	41.53	44.62	42.15	34.44	41.92	41.80
S_2	27.57	24.91	39.14	31.81	32.59	32.81	31.23	**50.00**
S_{21}	60.46	74.91	80.67	76.43	74.74	67.25	73.15	**91.80**

Table 7
The comparisons between the proposed algorithm and other algorithms for CEC2022.

	JADE	SHADE	LSHADE	FLDE	DFLDE	DEDQN	RLHPSDE	RLDEFLA
S_{10}	42636	152756	**160329**	105027	116481	104980	142668	123613
S_{20}	58346	124541	**163052**	109676	129605	118809	103161	147997
S_{22}	100982	277297	**323381**	214703	246086	223789	245829	271610

4.3.3. *Score Results for CEC2021 and CEC2022*

Tables 6 and 7 show the comparisons of the proposed algorithm (i.e. RLDEFLA) with other metaheuristic algorithms (i.e. JADE, SHADE, LSHADE), metaheuristic algorithms with FLA (i.e. FLDE and DFLDE), and metaheuristic algorithms with FLA and RL (i.e. DEDQN and RLHPSDE). The results of Table 6 show that the proposed algorithm can perform better than all the other metaheuristic algorithms compared in this study in terms of the score metric S_{21} for CEC2021 benchmarks. SHADE achieves better results than RLDEFLA on S_1, which considers the average normalized error across all functions, and the difference between the best result of a search algorithm and the best-known solution. However, RLDEFLA can significantly outperform SHADE in terms of S_2, which is the rank of an algorithm among all algorithms for all test functions. These comparison results show that RLDEFLA can provide a balanced way for solving the SOP.

Table 7 further shows that the proposed algorithm can still obtain better results than metaheuristic algorithms with FLA (i.e. FLDE and DFLDE), and metaheuristic algorithms with FLA and RL (i.e. DEDQN and RLHPSDE) in terms of S_{22}, but it fails to surpass SHADE and LSHADE. Based on our observations, we attribute this to two reasons. First, FLA-based mechanisms require additional evaluations, which will degrade the score when convergence speed and results are considered together. Second, the parameter settings were determined by TPE over the 50 CEC2021 and 12 CEC2022 benchmarks, so they are strongly influenced by CEC2021. We therefore reran TPE on the CEC2022 benchmarks alone to find suitable parameter settings. The results of S_{10}, S_{20}, and S_{22} for the proposed algorithm are 175142, 168445, and 343587, respectively, indicating that with suitable parameter settings, the proposed algorithm can achieve better results. These results are now better than SHADE and LSHADE, and significantly better than other FLA- and RL-based methods for metaheuristic algorithms, but it still has room for improvement to reduce the costs associated with the FLA.

4.3.4. *Average Rank and Wilcoxon Test Results for CEC2021 and CEC2022*

To compare the performance between RLDEFLA and other search algorithms, the average rank of each algorithm is shown in Fig. 7. Note that the average rank of an algorithm in

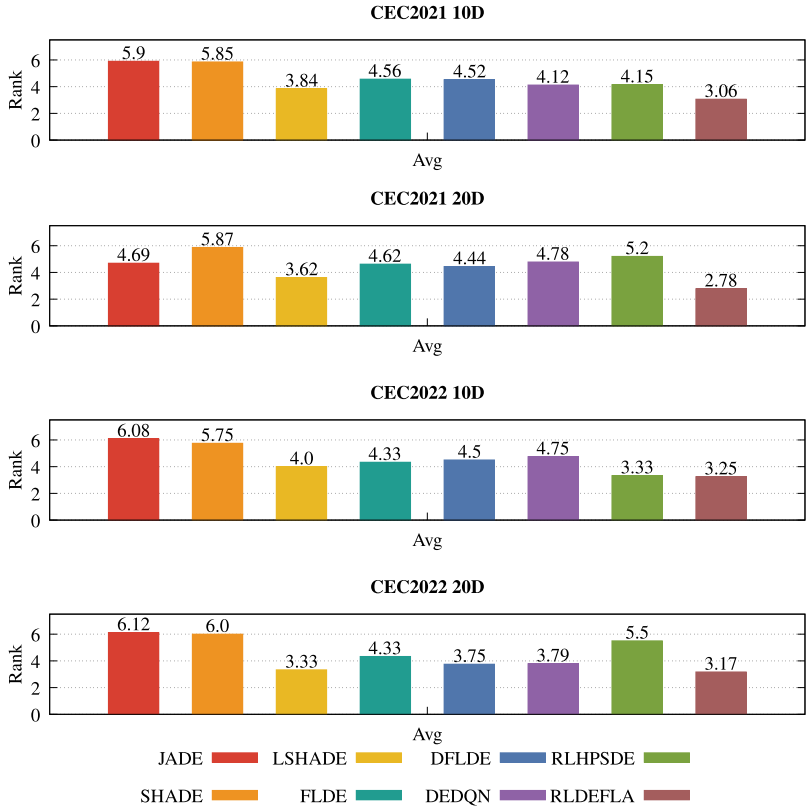


Fig. 7. The average rank of each algorithm for CEC2021 and CEC2022.

this figure represents its average rank among all algorithms for all given test functions in all trials of CEC2021 or CEC2022, respectively. Also note that the "Rank" represents the average rank value of 51 independent runs of all test functions of CEC2021 (with 10D and 20D) and CEC2022 (with 10D and 20D), respectively. The results indicate that RLDEFLA achieves better performance than other algorithms for both CEC2021 and CEC2022 in most cases. More precisely, in these comparisons, the rank with a lower value means the search algorithm can get better results on average in terms of the objective value. That RLDEFLA can get lowest rank values of 3.06 and 2.78 for CEC2021 with 10D and 20D and rank values of 3.25 and 3.17 for CEC2022 with 10D and 20D than other search algorithms, respectively, explains that the proposed algorithm can find better results than other state-of-the-art search algorithms (including the search algorithms based on FLA) in most cases.

To further understand the performance of the proposed method, statistical analysis is used for the comparisons between the RLDEFLA and the other search algorithms. Figure 8 shows the results of the Wilcoxon test (Wilcoxon, 1945) for these algorithms. The terms "Better," "Equal," and "Worse" indicate that the result of RLDEFLA is significantly better than, not significantly better than (i.e. similar), and worse than the compared

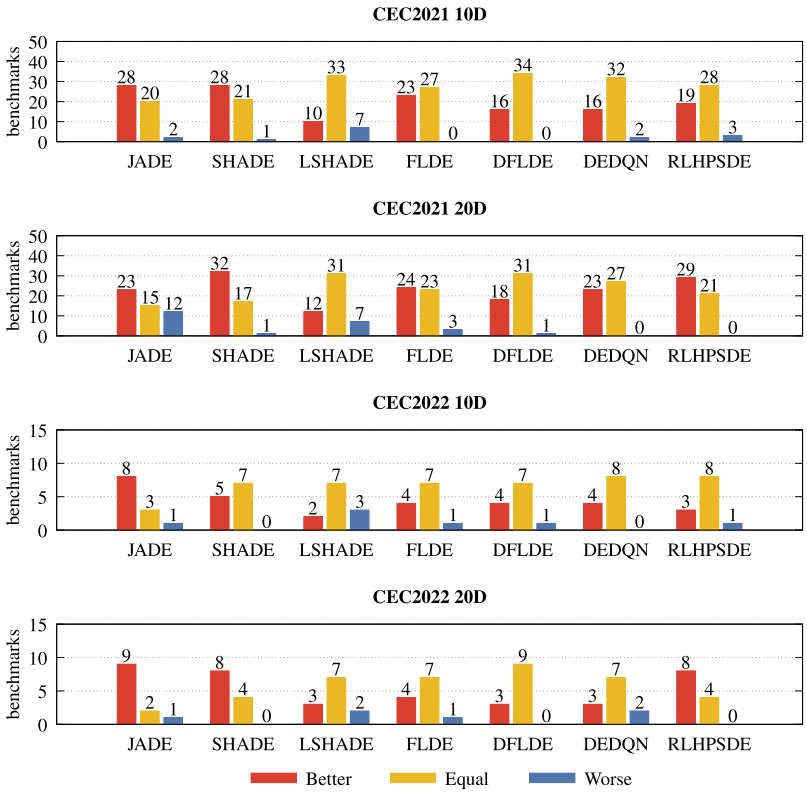


Fig. 8. The Wilcoxon test comparisons between RLDEFLA and the other algorithms for CEC2021 and CEC2022.

search algorithm, respectively. The results show that the proposed algorithm (RLDEFLA) outperforms other algorithms for both CEC2021 and CEC2022. As shown in Fig. 8, the proposed algorithm can find better results than the other seven search algorithms in most cases. For example, the results of CEC2021 with 10D show that the proposed algorithm can find better results than JADE in 28 test functions, similar results with JADE in 20 test functions, and worse results than JADE in only 2 test functions. Since the proposed algorithm is worse than JADE in 2 out of 50 test functions while the number of functions the proposed algorithm can get better results is more than the number of functions the proposed algorithm can get worse results (28 better vs. 2 worse), it can be easily seen that the proposed algorithm can find better results than JADE in most cases. Of course, similar situations can also be found when comparing the proposed algorithm with JADE, SHADE, LSHADE, FLDE, DFLDE, DEDQN, and RLHPSDE for CEC2021 with 10D and 20D.

Also shown in Fig. 8 are the comparisons between the proposed algorithm and the other seven search algorithms for the CEC2022 with 10D and 20D. Even though the number of test functions is not like that of CEC2021, which has 50, CEC2022 still has 12 test functions. From these comparisons, it can also be seen that the number of test functions the proposed algorithm can get better results than the other search algorithms is still more

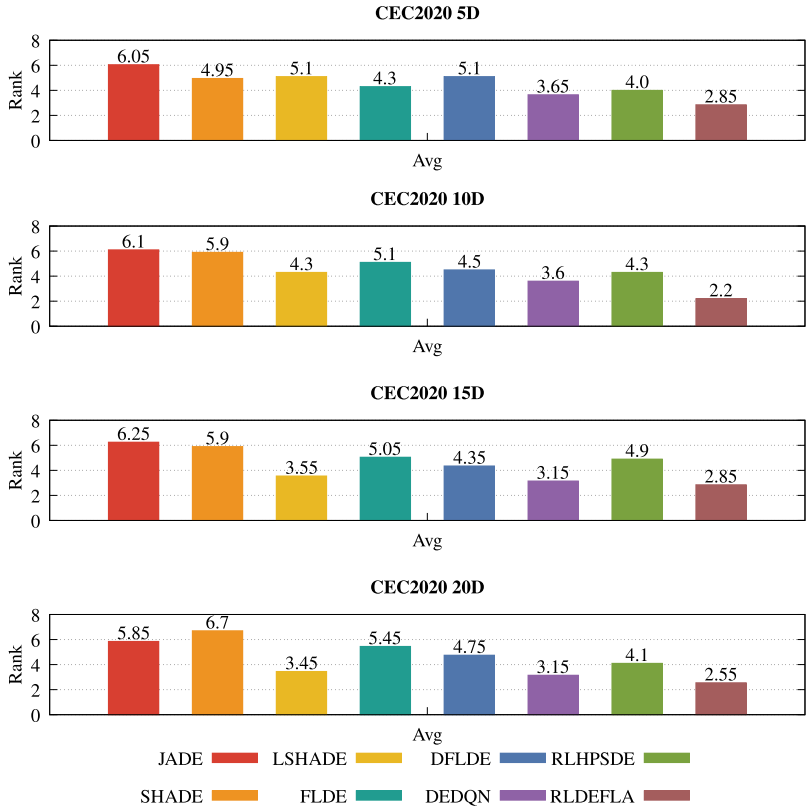


Fig. 9. The average rank of each algorithm for CEC2020.

than the number of functions the proposed algorithm only gets worse results like the comparisons in CEC2021. For example, the proposed algorithm can get better results in 8 test functions with 10D, similar results in 3 test functions, and worse results in only 1 test function in the comparisons between the proposed and JADE. In the other comparisons of Fig. 8, the proposed algorithm still can get better or similar results in most cases. It is worth noting that the proposed algorithm does not achieve better results on CEC2022 with 10D than LSHADE, even though the results are very similar. We believe that it is because of the parameter setting of the proposed algorithm, as mentioned in Section 4.3.3. As an alternative approach, based on the same procedure as in Section 4.3.3, we can further apply TPE solely to the CEC2022 benchmarks to determine suitable parameter settings. As a result, the proposed algorithm outperforms LSHADE on 3 test functions, performs comparably on 7, and performs worse on 2.

4.3.5. *Average Rank and Wilcoxon Test Results for CEC2020*

In this study, we further compare the proposed algorithm with other metaheuristic algorithms on the CEC2020 benchmark (Yue *et al.*, 2019) to assess its generalizability. Note that the maximum numbers of evaluations $E_{\max}$ for 5, 10, 15, and 20 dimensions are

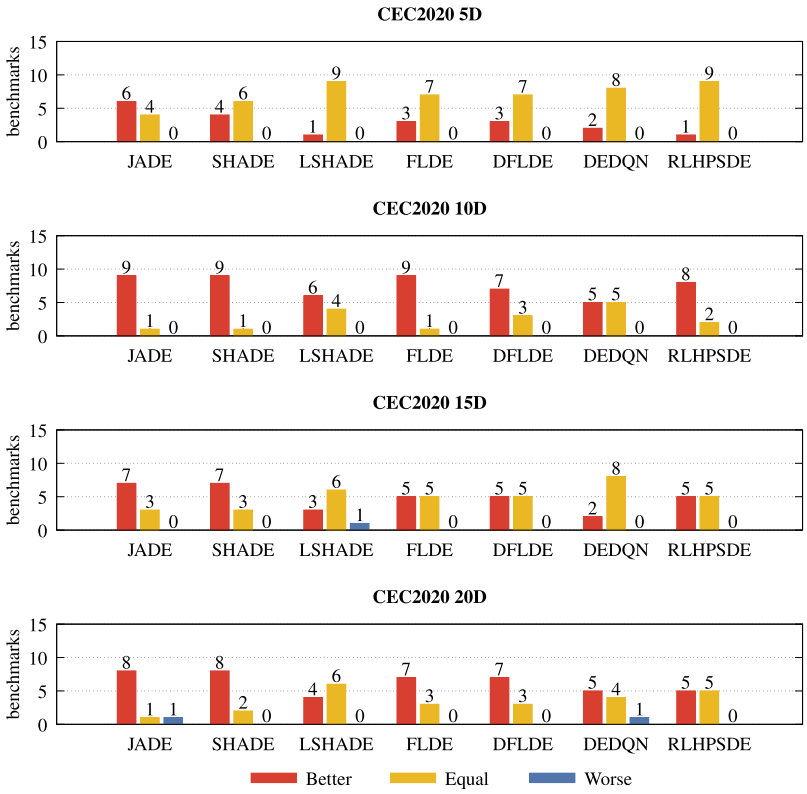


Fig. 10. The Wilcoxon test comparisons between RLDEFLA and the other algorithms for CEC2020.

50,000, 1,000,000, 3,000,000, and 10,000,000 for the CEC2020 according to the settings of Yue *et al.* (2019). As shown in Fig. 9, the proposed algorithm can still obtain better results than all the other metaheuristic algorithms for CEC2020, as measured by the metric from 5D to 20D.

Figure 10 further shows the comparison between the proposed algorithm and other metaheuristic algorithms for CEC2020 using the Wilcoxon test to determine whether the results of the proposed algorithm are better than those of the other metaheuristic algorithms. These results show that the proposed algorithm can find better results than all the other metaheuristic algorithms, demonstrating its generalizability for solving various SOPs. Note that from this comparison, although it can be easily seen that the proposed algorithm retains its advantage on the CEC2020 dataset without per-benchmark tuning, demonstrating that it has a good generalizability for CEC2020, CEC2021, and CEC2022, we cannot argue that the proposed algorithm will beat all the other algorithms compared in this study for all the other benchmarks to which it has not been applied. Instead, we can only argue that the proposed algorithm has a good generalizability for solving various SOPs, and it may provide better results than other search algorithms because the proposed algorithm embeds a good mechanism to automatically and dynamically pick a good tran-

sition operator from a set of operators to generate new candidate solutions during the convergence process. Based on these observations, we conclude that even if the proposed algorithm achieves good search performance on new optimization problems, it still needs to be modified to achieve optimal performance when the objectives, constraints, or input data distribution differ significantly from those of the test benchmarks used in this study.

5. Conclusions

This paper introduces a hybrid mutation operator for differential evolution based on the fitness landscape to improve search performance. The proposed algorithm represents an effective integration of supervised learning and unsupervised learning algorithms. The basic idea of the proposed algorithm is to first train a model using the known relationships between mutations and landscape characteristics. It can be viewed as a form of supervised learning. Then, this trained model is used for DE to adaptively determine the mutation operator for unseen single-objective optimization problems during the convergence process. It can be viewed as a form of unsupervised learning. The experimental results demonstrate that the proposed method improves the search performance on the CEC2020, CEC2021 and CEC2022 benchmarks, achieving better results than other DE-based algorithms for most functions. One of the important findings is that if we can have an accurate strategy detection model to dynamically choose the suitable search operator (e.g. mutation operator for DE) during the convergence process, the end results of the metaheuristic algorithm can then be significantly improved compared with those of a metaheuristic algorithm without such mechanism. Using a machine learning algorithm is a viable approach to generate such a strategy detection model for a metaheuristic algorithm. The comparison results of this study show that this can be a good solution to enhance the searchability of a metaheuristic algorithm. Further studies could investigate the potential of combining the proposed algorithm with other search algorithms. Also, future work could explore an effective alternative to the random walk for sampling fitness landscape characteristics.

Acknowledgements

The authors would like to thank the handling editor and anonymous reviewers for their valuable comments and suggestions on the paper.

Funding

National Science and Technology Council of Taiwan, ROC, under Contracts NSTC112-2628-E-110-001-MY3, NSTC114-2634-F-110-001-MBK, NSTC114-2221-E-110-023-MY3, and NSTC115-2221-E-110-039-MY3.

References

Abualigah, L., Elaziz, M.A., Khasawneh, A.M., Alshinwan, M., Ibrahim, R.A., Al-qaness, M.A.A., Mirjalili, S., Sumari, P., Gandomi, A.H. (2022). Meta-heuristic optimization algorithms for solving real-world mechanical engineering design problems: a comprehensive survey, applications, comparative analysis, and results. *Neural Computing and Applications*, 34, 4081–4110.

Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B. (2011). Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems*, 24.

Blum, C., Roli, A. (2003). Metaheuristics in combinatorial optimization: overview and conceptual comparison. *ACM Computing Surveys*, 35(3), 268–308.

da Costa Oliveira, A.L., Britto, A., Gusmão, R. (2023). Machine learning enhancing metaheuristics: a systematic review. *Soft Computing*, 27(21), 15971–15998.

Dokeroglu, T., Sevinc, E., Kucukyilmaz, T., Cosar, A. (2019). A survey on new generation metaheuristic algorithms. *Computers & Industrial Engineering*, 137, 106040.

Dorigo, M., Maniezzo, V., Colorni, A. (1991). *Ant System: An Autocatalytic Optimizing Process*. Technical report, Dipartimento di Elettronica, Politecnico di Milano, Italy. Technical Report 91-016. Available at https://citeseerx.ist.psu.edu/viewdoc/download?rep=rep1&type=pdf&doi=10.1.1.51.4214, Accessed 2022.08.01.

Elaziz, M.A., Dahou, A., Abualigah, L., Yu, L., Alshinwan, M., Khasawneh, A.M., Lu, S. (2021). Advanced metaheuristic optimization techniques in applications of deep neural networks: a review. *Neural Computing and Applications*, 33, 14079–14099.

Glover, F. (1989). Tabu search—Part I. *ORSA Journal on Computing*, 1(3), 190–206.

Glover, F., Kochenberger, G.A. (Eds.) (2003). *Handbook of Metaheuristics*. Kluwer Academic Publishers, Boston, MA.

Grefenstette, J.J. (1999). Evolvability in dynamic fitness landscapes: a genetic algorithm approach. In: *Proceedings of the Congress on Evolutionary Computation*, Vol. 3, pp. 2031–2038.

Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI.

Jones, T., Forrest, S. (1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In: *Proceedings of the International Conference on Genetic Algorithms*, pp. 184–192.

Kennedy, J., Eberhart, R.C. (1995). Particle swarm optimization. In: *Proceedings of the IEEE International Conference on Neural Networks*, pp. 1942–1948.

Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P. (1983). Optimization by Simulated Annealing. *Science*, 220(4598), 671–680.

Kumar, A., Price, K.V., Mohamed, A.W., Hadi, A.A., Suganthan, P.N. (2021). *Problem definitions and evaluation criteria for the CEC 2022 special session and competition on single objective bound constrained numerical optimization*. Technical report, Nanyang Technological University, Singapore. Available at https://github.com/P-N-Suganthan/2022-SO-BO, Accessed 2024.05.01.

Liang, J.J., Qu, B., Suganthan, P.N., Hernández-Díaz, A.G. (2013). *Problem definitions and evaluation criteria for the CEC 2013 special session on real-parameter optimization.* Technical report. Available at https://github.com/P-N-Suganthan/CEC2013, Accessed 2024.05.01.

Lin, M., Wang, Z., Zheng, W. (2023). Hybrid particle swarm-differential evolution algorithm and its engineering applications. *Soft Computing*, 27(22), 16983–17010.

Liu, Y., Sun, Y., Xue, B., Zhang, M., Yen, G.G., Tan, K.C. (2023). A survey on evolutionary neural architecture search. *IEEE Transactions on Neural Networks and Learning Systems*, 34(2), 550–570.

Malan, K.M., Engelbrecht, A.P. (2009). Quantifying ruggedness of continuous landscapes using entropy. In: *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1440–1447.

Malan, K.M., Engelbrecht, A.P. (2013). A survey of techniques for characterising fitness landscapes and some possible ways forward. *Information Sciences*, 241, 148–163.

Merz, P., Freisleben, B. (1999). *Fitness Landscapes and Memetic Algorithm Design*. New Ideas in Optimization. McGraw-Hill Ltd., UK, GBR, pp. 245–260.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M.A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529–533.

Mohamed, A.W., Hadi, A.A., Mohamed, A.K., Agrawal, P., Kumar, A., Suganthan, P.N. (2020). *Problem definitions and evaluation criteria for the CEC 2021 special session and competition on single objective bound*

constrained numerical optimization. Technical report, Nanyang Technological University, Singapore. Available at https://github.com/P-N-Suganthan/2021-SO-BCO, Accessed 2024.05.01.

Ochoa, G., Malan, K. (2019). Recent advances in fitness landscape analysis. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 1077–1094.

Osaba, E., Villar-Rodriguez, E., Del Ser, J., Nebro, A.J., Molina, D., LaTorre, A., Suganthan, P.N., Coello Coello, C.A., Herrera, F. (2021). A tutorial on the design, experimentation and application of metaheuristic algorithms to real-world optimization problems. *Swarm and Evolutionary Computation*, 64, 100888.

Pitzer, E., Affenzeller, M. (2012). A comprehensive survey on fitness landscape analysis. In: *Recent Advances in Intelligent Engineering Systems*, pp. 161–191.

Rani, R., Jain, S., Garg, H. (2024). A review of nature-inspired algorithms on single-objective optimization problems from 2019 to 2023. *Artificial Intelligence Review*, 57(126), 1–51.

Reidys, C.M., Stadler, P.F. (2001). Neutrality in fitness landscapes. *Applied Mathematics and Computation*, 117(2), 321–350.

Richter, H., Engelbrecht, A.P. (Eds.) (2014). *Recent Advances in the Theory and Application of Fitness Landscapes*. Springer, Berlin, Heidelberg.

Seyyedabbasi, A. (2023). A reinforcement learning-based metaheuristic algorithm for solving global optimization problems. *Advances in Engineering Software*, 178, 103411.

Sharma, V., Tripathi, A.K. (2022). A systematic review of meta-heuristic algorithms in IoT based application. *Array*, 14, 100164.

Shen, L., He, J. (2010). A mixed strategy for Evolutionary Programming based on local fitness landscape. In: *IEEE Congress on Evolutionary Computation*, pp. 1–8.

Soler-Dominguez, A., Juan, A.A., Kizys, R. (2017). A survey on financial applications of metaheuristics. *ACM Computing Surveys*, 50(1), 1–23.

Stadler, P.F. (2002). Fitness landscapes. In: Lässig, M., Valleriani, A. (Eds.), *Biological Evolution and Statistical Physics*. Springer, Berlin, Heidelberg, pp. 183–204.

Storn, R., Price, K. (1995). *Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces*. Technical report, International Computer Science Institute (ICSI). Technical Report TR-95-012, Available at https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.67.5398&rep=rep1&type=pdf, Accessed 2024.05.01.

Talbi, E.-G. (2021). Machine learning into metaheuristics: a survey and taxonomy. *ACM Computing Surveys*, 54(6), 1–32.

Tan, Z., Li, K. (2021). Differential evolution with mixed mutation strategy based on deep reinforcement learning. *Applied Soft Computing*, 111, 107678.

Tan, Z., Li, K., Wang, Y. (2021). Differential evolution with adaptive mutation strategy based on fitness landscape analysis. *Information Sciences*, 549, 142–163.

Tan, Z., Tang, Y., Li, K., Huang, H., Luo, S. (2022a). Differential evolution with hybrid parameters and mutation strategies based on reinforcement learning. *Swarm and Evolutionary Computation*, 75, 101194.

Tan, Z., Tang, Y., Huang, H., Luo, S. (2022b). Dynamic fitness landscape-based adaptive mutation strategy selection mechanism for differential evolution. *Information Sciences*, 607, 44–61.

Tanabe, R., Fukunaga, A. (2013). Success-history based parameter adaptation for differential evolution. In: *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 71–78.

Tanabe, R., Fukunaga, A.S. (2014). Improving the search performance of SHADE using linear population size reduction. In: *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1658–1665.

Tessari, M., Iacca, G. (2022). Reinforcement learning based adaptive metaheuristics. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 1854–1861.

Tong, H., Huang, C., Minku, L.L., Yao, X. (2021). Surrogate models in evolutionary single-objective optimization: a new taxonomy and experimental study. *Information Sciences*, 562, 414–437.

Tsai, C.W., Chiang, M.C. (2023). *Handbook of Metaheuristic Algorithms: From Fundamental Theories to Advanced Applications*. Elsevier.

Vassilev, V.K., Fogarty, T.C., Miller, J.F. (2000). Information characteristics and the structure of landscapes. *Evolutionary Computation*, 8(1), 31–60.

Velasco, L., Guerrero, H., Hospitaler, A. (2024). A literature review and critical analysis of metaheuristics recently developed. *Archives of Computational Methods in Engineering*, 31, 125–146.

Wang, M., Li, B., Zhang, G., Yao, X. (2018). Population evolvability: dynamic fitness landscape analysis for population-based metaheuristic algorithms. *IEEE Transactions on Evolutionary Computation*, 22(4), 550–563.

Weinberger, E.D. (1990). Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, 63, 325–336.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6), 80–83.

Wolpert, D.H., Macready, W.G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1), 67–82.

Wright, S. (1932). The roles of mutation, inbreeding, crossbreeding and selection in evolution. In: *Proceedings of the International Congress of Genetics*, Vol. 1, pp. 356–366.

Yaqoob, A., Verma, N.K., Aziz, R.M. (2024). *Metaheuristic algorithms and their applications in different fields*. Metaheuristics for Machine Learning: Algorithms and Applications. John Wiley & Sons, pp. 1–35.

Yue, C., Price, K.V., Suganthan, P.N., Liang, J., Ali, M.Z., Qu, B., Awad, N.H., Biswas, P.P. (2019). *Problem definitions and evaluation criteria for the CEC 2020 special session and competition on single objective bound constrained numerical optimization*. Technical report, Zhengzhou University, China. Available at https://github.com/P-N-Suganthan/2020-Bound-Constrained-Opt-Benchmark, Accessed 2024.05.01.

Zhang, J., Sanderson, A.C. (2009). JADE: adaptive differential evolution with optional external archive. *IEEE Transactions on Evolutionary Computation*, 13(5), 945–958.

Zhang, L., Lu, G., Yan, X., Xia, P., Chen, Z., Wu, D. (2025). A differential evolution optimized hybrid XGBoost for accurate carbon emission prediction. *Environmental Modelling & Software*, 193, 106627.

Zhang, Z., Wu, Z., Zhang, H., Wang, J. (2023). Meta-learning-based deep reinforcement learning for multiobjective optimization problems. *IEEE Transactions on Neural Networks and Learning Systems*, 34(10), 7978–7991.

Zhou, X., Song, J., Wu, S., Wang, M. (2023). Artificial bee colony algorithm based on online fitness landscape analysis. *Information Sciences*, 619, 603–629.

Zou, F., Chen, D., Liu, H., Cao, S., Ji, X., Zhang, Y. (2022). A survey of fitness landscape analysis for optimization. *Neurocomputing*, 503, 129–139.

C.-W. Tsai received his PhD degree in computer science and engineering from National Sun Yat-sen University, Kaohsiung, Taiwan, in 2009. He is currently an associate professor and the associate vice president for Library and Information Services at National Sun Yat-sen University. He has also been a joint-appointment associate professor with the College of Semiconductor and Advanced Technology Research at National Sun Yat-sen University and a global joint-appointment professor at Kyungpook National University, Daegu, Korea, since 2024 and 2026, respectively. His research interests include computational intelligence, cloud computing, and the Internet of Things.

C.-C. Chen received his BS degree in computer science and information engineering from National Chi Nan University, Nantou, Taiwan, in 2022, and his MS degree in Computer Science and Engineering from National Sun Yat-sen University, Kaohsiung, Taiwan, in 2024. His research interests include metaheuristic algorithms, machine learning, and single-objective optimization.

K.-H. Chen received his BS degree in aeronautics and astronautics from National Cheng Kung University, Tainan, Taiwan, in 2024, and his MS degree in computer science and engineering from National Sun Yat-sen University, Kaohsiung, Taiwan, in 2026. His research interests include metaheuristic algorithms, reinforcement learning, fitness landscape analysis, and single-objective optimization.

R. Gipiškis received his PhD in computer science from Vilnius University in 2025. He is currently a researcher at the Institute of Data Science and Digital Technologies (Vilnius University) and a research analyst at AI Standards Lab, a non-profit focused on technical AI safety standards. His research interests include model interpretability, evaluation, risk management, and technical AI governance.

O. Kurasova received a PhD degree in computer science from the Institute of Mathematics and Informatics, Vytautas Magnus University (Lithuania), in 2005. She is currently employed as a principal researcher and a professor at the Institute of Data Science and Digital Technologies, Vilnius University (Lithuania). Her research interests include data mining methods, optimization theory and applications, artificial intelligence, neural networks, visualization of multidimensional data, multiple criteria decision support, parallel computing, and image processing. She is the author of more than 100 scientific publications.