SN Computer Science (2024) 5:771
https://doi.org/10.1007/s42979-024-03062-2

ORIGINAL RESEARCH

Dynamic Mutation Strategy Selection in Differential Evolution Using Perturbed Adaptive Pursuit

Prathu Bajpai[1] · Ogbonnaya Anicho[2] · Atulya K. Nagar[2] · Jagdish Chand Bansal[1]

Received: 21 March 2024 / Accepted: 17 June 2024

Abstract

Diverse mutant vectors play a significant role in the performance of the Differential Evolution (DE). A mutant vector is generated using a stochastic mathematical equation, known as mutation strategy. Many mutation strategies have been proposed in the literature. Utilizing multiple mutation strategies with the help of an adaptive operator selection (AOS) technique can improve the quality of the mutant vector. In this research, one popular AOS technique known as perturbation adaptive pursuit (PAP) is integrated with the DE algorithm for managing a pool of mutation strategies. A community-based reward criterion is proposed that rewards the cumulative performance of the whole population. The proposed approach is called '*Dynamic Mutation Strategy Selection in Differential Evolution using Perturbed Adaptive Pursuit (dmss-DE-pap)*'. The performance of dmss-DE-pap is evaluated over the 30D and 50D optimization problems of the CEC 2014 benchmark test suite. Results are competitive when compared with other state-of-the-art evolutionary algorithms and some recent DE variants.

Keywords Differential evolution · Evolutionary optimization · Meta-heuristics · Adaptive pursuit strategy · Mutations

Mathematics Subject Classification 68W50 · 68T05 · 68T20 · 90C59

Introduction

Recently, the popularity of population-based evolutionary algorithms (EAs) has grown extensively due to their robust capabilities in solving complex real-world optimization problems [1]. The optimization problems arise across various disciplines of science and engineering, including domains like data mining, machine learning, and artificial intelligence [2, 3]. The objective function(s) associated with such optimization problems are defined over high-dimensional search spaces and are highly non-linear, non-convex, and non-differentiable. The lack of gradient-specific information coupled with high computational cost makes many traditional optimization techniques obsolete or difficult to use [4]. Evolutionary algorithms (EAs) are modern stochastic optimization techniques that follow a non-conventional gradient-free approach. These algorithms do not assume any specific properties like linearity, continuity, or convexity regarding the underlying objective function(s) and follow the principles of minimal information availability [5]. However, due to their stochastic search behavior, these algorithms are prone to stuck in the regions of local optimal solution(s) and may suffer stagnation or premature convergence [6, 7].

The Differential Evolution (DE) algorithm is a popular population-based evolutionary algorithm [8]. It involves two stochastic operations, namely mutation and crossover, and three control parameters: population size (N), scaling factor (F), and crossover rate (CR). First, a population of N representative solutions (known as vectors in DE terminology) is randomly initialized in the search space, and then new

Ogbonnaya Anicho, Atulya K. Nagar and Jagdish Chand Bansal have contributed equally to this work.

✉ Atulya K. Nagar
atulya.nagar@hope.ac.uk

Prathu Bajpai
prathubajpai1812@gmail.com

Ogbonnaya Anicho
anichoo@hope.ac.uk

Jagdish Chand Bansal
jcbansal@sau.ac.in

[1] Department of Mathematics, South Asian University, Rajpur Road, New Delhi, Delhi 110068, India

[2] Faculty of Science, Liverpool Hope University, Hope Park, Liverpool L16 9JD, UK

Published online: 08 August 2024

solutions are generated using stochastic mathematical equations. These stochastic mathematical equations are referred to as mutation strategies in the DE literature and are used to define the DE mutation operator [9–11]. It has been studied that the performance of the DE algorithm is significantly affected by the choice of mutation strategy defining the mutation operator [12, 13]. The inappropriate choice of mutation strategy may lead to premature convergence in the DE algorithm [13–15].

To improve this performance issue, the utilization of multiple mutation strategies with adaptive control parameter settings in a single run of the DE optimization procedure was first proposed in the SaDE algorithm [16]. Then, following similar research direction, different DE variants, like EPSDE [17], CoDE [18], MPEDE [19], EDEV [20], MMRDE [21], and DISDE [22] were proposed that utilize multiple mutation strategies. However, it is interesting to note that the idea of utilizing multiple mutation strategies in an online manner in the DE algorithm can be seen as equivalent to implementing the adaptive operator selection (AOS) technique for selecting various mutation strategies. Three major research questions may appear while utilizing multiple mutation strategies. The first question is how many mutation strategies should be utilized, the second question is what type of mutation strategies should be utilized, and the third question is how to utilize them for optimal performance. This research focuses on studying the third question that is how to utilize multiple mutation strategies in order to obtain optimal performance. Major contributions of this research work are as follows.

- An adaptive operator selection (AOS) technique, known as the perturbation adaptive pursuit (PAP) strategy, is integrated with DE for utilizing multiple mutation strategies in a single run.
- A community-based reward criterion is proposed in which a selected mutation strategy is credited with a positive or negative reward based on its cumulative performance on the population.
- For saving the computationally intensive task of manual parameter tuning, the success-history-based parameter adaption technique is employed for adapting scalar factor F, and crossover rate CR during the optimization procedure. A linear population reduction scheme is utilized for adapting the population size.
- Experiments are conducted on CEC 2014 benchmark problems and experimental results are compared with some other evolutionary algorithms and DE variants utilizing multiple mutation strategies for a fair comparison.

The rest of the paper is organized as follows: Background for this research is given in section "Background". A brief discussion on the perturbed adaptive pursuit (PAP) is given in section "The Adaptive Pursuit Strategy". The detailed discussion on the proposed dmss-DE-pap algorithm is given in section "Dynamic Mutation Strategy Selection in Differential Evolution Using Perturbed Adaptive Pursuit (dmss-DE-pap)". Experimental results are given in section "Experimental Results" and a discussion of the obtained results is given in section "Discussion". Section "Conclusion" concludes the research.

Background

In this section, first, the working procedure of the canonical DE is discussed. Then a brief review of DE variants incorporating multiple mutation strategies is given. Since the idea of utilizing multiple mutation strategies in a single run of the optimization procedure is similar to the technique of adaptive operator selection (AOS), some relevant AOS techniques are also discussed for the sake of completeness.

Canonical Differential Evolution Algorithm

The Differential Evolution (DE) algorithm is a population-based stochastic optimizer [8]. The working procedure of the DE algorithm involves two major evolutionary operations, known as mutation and crossover, and three major control parameters namely *population size* N, *scaling factor* F, and *crossover rate* CR. For initialization, a set $P = \{X_1, X_2, \ldots, X_N\}$, is initialized. The elements of P represent uniformly distributed random solutions in the bounded D-dimensional search space. The vector $X_i^t = \{x_{i,1}, x_{i,2}, \ldots, x_{i,D}\}, i = 1, 2, \ldots, N$. represents the state or position of the i^{th} solution in the search space, at particular time instance t, where t denotes the iteration counter. This population P is evolved iteratively to approximate the global optimal solution(s). The algorithmic procedure of the DE algorithm is illustrated in Algorithm 1. The DE algorithm is terminated when either the desired accuracy of the approximation is achieved or the maximum number of iterations t_{max} is exhausted.

Algorithm 1 Differential Evolution Algorithm

Require: Objective Function f, Dimension D, Search Bounds $[x_{min_j}, x_{max_j}]$, $1 \leq j \leq D$, Pop Size N, Scaling Factor F, Crossover Rate CR. Iteration counter t
Ensure: Global Optimum $f(X_{best})$, Global Optimal Solution X_{best}
Set iteration counter $t \leftarrow 0$
Initialize $X_i^0 = (x_{i,1}, \ldots, x_{i,D})$ for each $1 \leq i \leq N$

$$x_{i,j} = x_{min_j} + rand(0,1) * (x_{max_j} - x_{min_j})$$

while termination criteria not satisfied **do**
  for $i \leftarrow 0; i < N; i++$ **do**
    Choose r_1, r_2, r_3 randomly s.t. r_1, r_2, r_3 and i all are distinct // Mutation Phase

$$V_i^{t+1} \leftarrow X_{r_1}^t + F \cdot (X_{r_2}^t - X_{r_3}^t) \tag{1}$$

    // $X_i^t = (x_{i,1}, \ldots, x_{i,D})$ is called target vector and $V_i^{t+1} = (v_{i,1}, \ldots, v_{i,D})$ is called trial vector, $1 \leq i \leq N$.
    Pick random dimension $j_{rand} \leftarrow rand\{1, D\}$ // Crossover Phase
    for $j \leftarrow 0, j < D, j++$ **do**
      if $rand(0,1) < CR$ or $j = j_{rand}$ **then**
        $u_{i,j} \leftarrow v_{i,j}$
      else
        $u_{i,j} \leftarrow x_{i,j}$
      end if
    // $U_i^t = (u_{i,1}, \ldots, u_{i,D})$ is called child vector, $1 \leq i \leq N$
    end for
    // Selection Phase (Minimization problems)
    if $f(U_i^t) < f(X_i^t)$ **then**
      $X_i^{t+1} \leftarrow U_i^t$
    else
      $X_i^{t+1} \leftarrow X_i^t$
    end if
  end for
  $t \leftarrow t + 1$
end while

The equation 1 in the Algorithm 1 is referred to as mutation operator of the DE. Many mutation strategies have been proposed in the DE literature, some of the widely used mutation strategies are listed below:

- **DE/rand/1**

$$V_i^{t+1} = X_{r_1}^t + F \cdot (X_{r_2}^t - X_{r_3}^t) \tag{2}$$

- **DE/best/1**

$$V_i^{t+1} = X_{best}^t + F \cdot (X_{r_2}^t - X_{r_3}^t) \tag{3}$$

- **DE/rand/2**

$$V_i^{t+1} = X_{r_1}^t + F \cdot (X_{r_2}^t - X_{r_3}^t) + F \cdot (X_{r_4}^t - X_{r_5}^t) \tag{4}$$

- **DE/best/2**

$$V_i^{t+1} = X_{best}^t + F \cdot (X_{r_1}^t - X_{r_2}^t) + F \cdot (X_{r_3}^t - X_{r_4}^t) \tag{5}$$

- **DE/current-to-pbest/1**

$$V_i^{t+1} = X_i^t + F \cdot (X_{pbest}^t - X_i^t) + F \cdot (X_{r_1}^t - X_{r_2}^t) \tag{6}$$

Here, $r_k, 1 \leq k \leq 5$ are randomly selected indices distinct from i in the range [1, N]. $X_{r_k}^t$ and V_i^{t+1} represent target vectors and trial vector, respectively. X_{best}^t in equations (3), (5), is the best vector and, X_{pbest}^t in equation (6) is a randomly selected vector from the top $p\%$ vectors in the current population.

It has been shown in previous works that the performance of these mutation strategies largely depends on the underlying functional landscape [13, 14, 16]. In an attempt to mitigate this dependence, the idea of utilizing multiple mutation strategies is explored in the DE literature. Several successful DE variants have been proposed. In the next subsection, a discussion on some of such popular DE variants utilizing multiple mutation strategies is given.

DE Variants with Multiple Mutation Strategies

Mutation strategies in the DE algorithm utilize a weighted vector difference approach for generating trial vectors in the search space. In general, mutation strategies utilize one vector difference or two vector difference based scaling to generate mutant vector. However, the performance of these mutation strategies is affected by the characteristics of the underlying objective function(s) [23]. For instance, mutation strategies involving two vector differences such as 'DE/rand/2' and 'DE/best/2' produce more diverse vectors than strategies involving single vector differences such as 'DE/rand/1' and 'DE/best/1'. The mutation strategies involving single vector differences perform better on unimodal problems, while their performance suffers badly on multi-modal problems [24]. Hence, for solving black-box optimization problems, it is desirable that modality of the function do not limit the search scope of the DE algorithm.

Inspired by this, Qin et al. [16] proposed the SaDE algorithm that utilized multiple mutation strategies in the optimization procedure of the DE algorithm. Mallipeddi et al. [17] proposed the EPSDE algorithm in which a pool of three distinct mutation strategies was used. Similarly, Wang et al. [18] proposed the CoDE algorithm in which three distinct mutation strategies were used along with three different settings of control parameters F, and CR. Wu et al. [19] proposed the MPEDE algorithm in which three distinct mutation strategies were used with a multi-population framework, and competitive results were reported. The same authors Wu et al. [20] proposed the EDEV algorithm in which an ensemble of three different DE variants, namely JADE, CoDE, and EPSDE were utilized in a single run of the optimization procedure and compared with other state-of-the-art DE variants.

However, the idea of utilizing multiple mutation strategies as an adaptive operator selection (AOS) technique was first studied in the context of operator selection in genetic algorithm [25]. Inspired by this, Gong et al. [26] conducted an empirical study on adaptive strategy selection in the DE algorithm for numerical optimization problems. Two adaptive operator selection techniques, namely, probability matching [27], and adaptive pursuit [25], were integrated into the JADE algorithm with a pool containing 4 mutation strategies of the DE algorithm. For assigning credits to a particular mutation strategy, a credit assignment technique based on the fitness values of the target and test vectors was proposed. However, the major drawback associated with such credit assignment is that a particular mutation strategy may accumulate excessive credits (or, rewards) at an early phase of the evolutionary process, which in turn depletes the role of other available strategies in the strategy pool [28]. Considering these factors into account, in this research, a novel community-based reward criterion is proposed. The underlying population is treated as a community, and a particular mutation strategy is rewarded if more than $(100 \cdot \alpha)\%$, where $\alpha = 0.6$, mutant vectors are better than their corresponding target vectors. This ensures that the algorithm has less leverage in switching the mutation strategies compared to a pure greedy strategy selection approach based on the relative fitness of the individual. Further, instead of utilizing the adaptive pursuit (AP) strategy, a perturbation parameter-based adaptive pursuit strategy (PAPS) as proposed in [28] is integrated for mutation strategy selection. This modification ensures that the proposed algorithm readjusts the weights of the underlying mutation strategies in order to efficiently utilize the diversity of the operator pool. Moreover, the proposed method can be well utilized with other frameworks of the evolutionary algorithms, and not limited to the DE algorithm only. In the next section, a brief discussion of the adaptive pursuit strategy is given.

The Adaptive Pursuit Strategy

The Adaptive Pursuit Strategy is an adaptive technique for operator selection [25]. The adaptive pursuit comes under a broader class of rapidly converging algorithms for learning automata. These techniques adapt the probability vector of the operator in a manner that the algorithm chooses the operator with an estimated maximal reward. The adaptive pursuit strategies perform two major tasks of credit assignment and operator selection. Credit assignment is used to assign rewards to an operator based on its past performance in the search process, and operator selection is performed to choose an operator automatically for the future iterations in an iterative process [29]. Mathematically, it is defined as follows:

Suppose H is a set of K operators, say, $H = \{h_1, h_2, \ldots, h_K\}$.[1] For a given iteration t, let us assume, the associated selection probability vector (or, weight vector) is $W_t = \{w_{1,t}, w_{2,t}, \ldots, w_{K,t}\}$ where, $0 \leq w_{it} \leq 1 \;\; \forall i = 1, 2, \ldots, K$. The quality vector associated with the pool of operators is $Q_t = \{q_{1,t}, q_{2,t}, \ldots, q_{Kt}\}$ where,

[1] For this study, operators are different mutation strategies of the DE algorithm.

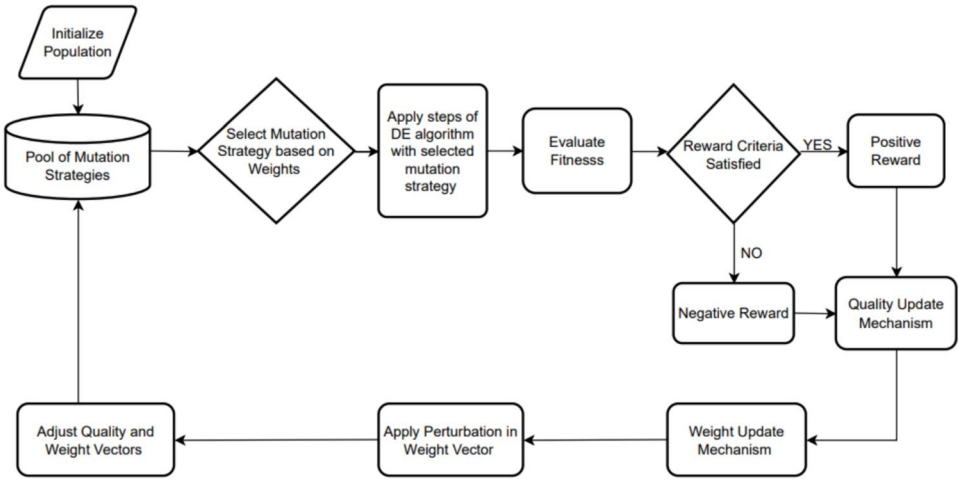


Fig. 1 Flow Chart of the dmss-DE-pap algorithm

$q_{i,0} = 1, \forall i = 1, 2, \ldots, K$. Say, for a selected operator h_i, w_{it} and q_{it} represent the weight and the quality values, respectively. The adaptive pursuit strategy is given as follows.

1. For a given iteration t, select an operator $h_{k,t}$, $1 \leq k \leq K$, from the pool of operators H in proportion to the selection probability vector $W_{k,t}$ and evaluate the reward $r_{k,t}$ for the operator $h_{k,t}$.
2. Update the quality value $q_{k,t}$ of $h_{k,t}$ based on the reward value $r_{k,t}$. Reward value can be positive or negative based on the performance of the underlying operator $h_{k,t}$.
3. If the quality value $q_{k,t}$ of $h_{k,t}$ improved, increase the selection probability of $h_{k,t}$ and decrease that of others, or vice-versa.
4. If the termination criteria are not satisfied, the adaptive pursuit strategy goes to step 1 for selecting an operator based on W_{t+1} for the next iterations.

The adaptive pursuit strategy helps in choosing the optimal operator from the pool of operators H automatically. However, this strategy is more suited for cases when the underlying environment is static. For dynamic environments like the evolving phase of an evolutionary algorithm, this strategy may favor strategies that perform better at the initial phase of the optimization procedure by assigning excessive rewards. Effectively, this will deplete the selection chances of other available strategies in the long run and may not utilize the pool diversity efficiently. Inspired by this, in this proposed work, an adaptive pursuit strategy with a perturbation parameter (PAPS) is utilized for strategy selection. A concise overview of PAPS is given in the next subsection.

Perturbation Adaptive Pursuit Strategy

The major drawback associated with the adaptive pursuit strategy for strategy selection in the context of the DE algorithm is that if a particular strategy performs better at an early phase of the evolutionary process, it may accumulate excessive rewards and dominate other strategies. This may lead to less-than-optimal utilization of alternative mutation strategies, and the full potential of the pool's diversity would not be realized [28]. However, if a perturbation parameter P_p is incorporated for adjusting the weights of underlying strategies. It can increase the chances of utilizing all other available strategies in the pool. Inspired by this, weights of strategies in the pool are randomly assigned when the value of a generated random number is less than the value of perturbation parameter P_p. This increases the robustness of the proposed dmss-DE-pap algorithm in strategy selection. A detailed description of the proposed algorithm is given in the next section.

Dynamic Mutation Strategy Selection in Differential Evolution Using Perturbed Adaptive Pursuit (dmss-DE-pap)

The proposed dmss-DE-pap algorithm is an integration of the perturbed adaptive pursuit for managing a pool of multiple mutation strategies in DE.

A pool of five different mutation strategies is created, and equal weights and quality values are assigned to each strategy. For initialization, a strategy is picked randomly from the pool, and the algorithmic procedure of the DE algorithm with a self-adaptive control parameter setting is executed. Based on the performance of the selected strategy, a positive or negative reward is allocated, and with the help of the received reward values, the quality and weight values are adjusted. This process is repeated iteratively till the termination criteria are not satisfied. The pseudo-code and flow chart of the proposed dmss-DE-pap algorithm is given in Algorithm 2, and Fig. 1, respectively. In the next subsections, a detailed description of components associated with the proposed dmss-DE-pap algorithm is given.

Algorithm 2 Dynamic Mutation Strategy Selection in Differential Evolution using Perturbed Adaptive Pursuit (dmss-DE-pap)

Require: Obj. Fun. f, Dim. D, Search Bounds $[x_{min_j}, x_{max_j}]$, $1 \leq j \leq D$, Pop Size N, Scaling Factor F, Crossover rate CR. Max_{iter}, Pool Size $K \leftarrow 5$, Reward Control Parameter $\alpha \leftarrow 0.6$, Quality Control Parameter $\beta \leftarrow 0.1$, Perturbation Probability $P_p \leftarrow 0.1$.
Ensure: Global Optimum $f(X_{best})$, Global Optimal Solution X_{best}
Set iteration counter $t \leftarrow 0$
Initialize strategy pool $H = \{h_1, h_2, \ldots, h_K\}$. h_k is DE mutation strategy, $k = 1, 2, \ldots, K$.
Initialize weight vector $W = (0.2, 0.2, \ldots, 0.2)$, Quality vector $Q = (1, 1, \ldots, 1)$, Reward Vector $R = (0, 0, \ldots, 0)$ s.t., $|W| = |Q| = |R| = K$.
Initialize local counter $LC = (lc_1, lc_2, \ldots, lc_K)$, Corresponding to each strategy h_k
while $t < Max_{iter}$ **do**
    Select a mutation strategy $h_k = \underset{1 \leq k \leq K}{argmax}(W)$
    Apply DE with selected mutation strategy h_k // (see Algorithm 1)
    for $i \leftarrow 0$; $i < NP$; $i++$ **do**
        if $f(U_i) < f(X_i)$ **then**
            $lc_k \leftarrow lc_k + 1$ // X_i : Parent Vector, U_i : Child Vector
        else
            $lc_k \leftarrow lc_k$
        end if
    end for
    Apply Reward-Assigning Mechanism // (see Algorithm 3)
    Apply Quality-Update Mechanism // (see Algorithm 4)
    Apply Perturbation-based Weight Update Mechanism // (see Algorithm 5)
    $t \leftarrow t + 1$
end while

Strategy Pool

The proposed dmss-DE-pap algorithm utilizes a pool containing five different mutation strategies of the DE algorithm as mentioned in Eqs. 2-6) at section "Canonical Differential Evolution Algorithm". The reason for selecting these five mutation strategies is their distinguished characteristics and performance in different kinds of optimization problems. For instance, strategies like 'DE/rand/2' and 'DE/best/2' are more suitable for multi-modal problems, and are more exploratory, however, these strategies provide slow convergence [21]. Similarly, strategies like 'DE/rand/1' and 'DE/best/1' have fast convergence speeds, and are better suited for solving global optimization problems, even, if they are prone to get stuck in the region of local optima [21]. 'DE/current-to-pbest/1' is an advanced mutation strategy, proposed in the JADE algorithm. There are two versions available for this strategy, one is with an external archive of top $p\%$ solutions, and the other is without an external archive, in which top $p\%$ individuals are chosen from the current population. This strategy is capable of producing more diverse vectors and has shown improved performance [30].

To incorporate all these characteristics in the operator pool, the proposed dmss-DE-pap algorithm utilizes the above-mentioned five strategies and selects a particular strategy based on its quality value and weight vector. The quality value of a selected mutation strategy is updated using the obtained reward values. In the next subsection, details about the reward-assigning mechanism are given.

Reward-Assigning Mechanism

The reward values serve as the feedback signals for updating the quality values of the underlying strategies. In the proposed dmss-DE-pap algorithm, a community-based reward-assigning mechanism has been proposed in which a strategy is credited with a positive reward value if more than $100 \cdot \alpha\%$, (where $\alpha = 0.6$), mutant vectors are better than their corresponding target vectors. This means a particular strategy is credited with a positive reward if it improves at least 60% of the current population, otherwise, the strategy is credited with a negative reward. The parameter α is called the reward control parameter, and the value of α near 1 can be considered an optimistic reward assignment, while

a value of α near 0 can be thought of as a pessimistic reward assignment. The major advantage of using this reward criterion is that the algorithm will not switch between strategies aggressively, and characteristics of the underlying strategy will be exploited till it stops improving the majority of the population. A local counter lc_k is used to record the instances where the mutant vectors are better than their corresponding target vectors while employing the k^{th} mutation strategy. The pseudo-code of the proposed reward-assigning mechanism is given in Algorithm 3.

Algorithm 3 Reward-Assigning Mechanism

Require: local counter $LC = (lc_1, lc_2, \ldots, lc_K)$, Pop Size N, reward control parameter α, Reward Vector $R = (r_{1,t}, r_{2,t}, \ldots, r_{K,t})$, iteration counter t.
Ensure: Reward $r_{k,t}$
at iteration t, if selected strategy is $h_{k,t}$,
if $lc_k \geq round(\alpha * N)$ **then**
    $r_{k,t} \leftarrow 1$
else
    $r_{k,t} \leftarrow -1$
end if

The proposed reward-assigning mechanism is different from the reward-assigning mechanisms proposed in [26], in the sense that, earlier proposed reward criterion was based on the individual's relative fitness values while the proposed reward-assigning mechanism in the dmss-DE-pap algorithm is based on the success of whole population or community. This ensures proper exploitation of the underlying mutation strategies in the strategy pool. The reward values obtained by a particular strategy will be utilized to update the quality values associated with underlying strategies. In the next section, details about the quality update mechanism are given.

Quality-Update Mechanism

Based on the received rewards by the underlying mutation strategies, the quality values stored in a quality vector Q are updated. Initially, all the mutation strategies in the pool are treated as of the same quality, and equal quality values are assigned, i.e., $Q = (1, 1, \ldots, 1)$. The quantum of the quality values for the underlying mutation strategies are updated using a quality control parameter β. In the proposed work, the value of β is taken to be 0.1. If a selected mutation strategy obtains a positive reward, then the associated quality value of that strategy is updated using the quality control parameter β, while keeping the quality of other mutation strategies unchanged. Similarly, if a selected mutation strategy receives a negative reward, the quality value associated with it is degraded while keeping the quality values of other mutation strategies unchanged. The pseudo-code for the quality-update mechanism is given in Algorithm 4.

Algorithm 4 Quality-Update Mechanism

Require: selected mutation strategy $h_{k,t}$, reward-value $r_{k,t}$, quality value $q_{k,t}$, quality control parameter β, iteration counter t.
Ensure: Quality value $q_{k,t+1}$
at iteration t, if selected strategy is $h_{k,t}$,
if $r_{k,t} > 0$ **then**
    $q_{k,t+1} \leftarrow (1 + \beta) * q_{k,t}$
else
    $q_{k,t+1} \leftarrow (1 - \beta) * q_{k,t}$
end if

Table 1 Initial parameter settings for all considered algorithms

Algorithm	Initial Parameters Settings
PSO [35]	$N = 100, C_1 = 2, C_2 = 2$, Topology: Fully Connected
DE [8]	$N = 100, F = 0.5$ and $CR = 0.7$
CMA-ES [36]	$N = 100, \mu = PS/2\ w = \log(\mu + 0.5) - \log(1 : \mu), \mu_{eff} = \frac{1}{w^2}$ $C_\sigma = \frac{\mu_{eff}+2}{D+\mu_{eff}+5}\ D_\sigma = 1 + C_\sigma + 2 * max(\sqrt{(\frac{\mu_{eff}-1}{D+1})} - 1, 0)$
ABC [34]	$N = 100$, Limit (L) $= 0.6 \cdot D \cdot PS, \phi = rand(-a, a), -1 \leq a \leq 1$ $N_{max} = 18 \cdot D, N_{min} = 4, \mu_F = 0.5$
LSHADE [31]	$F \sim C(\mu_F, 0.1), \mu_{CR} = 0.5$ $CR \sim C(\mu_{CR}, 0.1), p = 0.11$ $N_{max} = 12 \cdot D, N_{min} = 4$
i-LSHADE [37]	H, F and CR same as LSHADE, $\mu_F = 0.8$ $\mu_{CR} = 0.5, \mu_{F_H} = \mu_{CR_H} = 0.9$
jSO [38]	$N_{max} = 25 \cdot \ln D \cdot \sqrt{D}, N_{min} = 4, F$ and CR same as LSHADE, $\mu_F = 0.3$
SaDE [16]	$N = 100$, Operator Pool size $H = 4, F \in N[0.5, 0.3], CR \in N[0.5, 0.1]$
CoDE [18]	$N = 100$, Operator Pool size $H = 3, [F = 1.0, CR = 0.1], [F = 1.0, CR = 0.9]$ $[F = 0.8, CR = 0.2]$
EPSDE [17]	$N = 100$, Operator Pool size $H = 4, F = [0.4 : 0.1 : 0.9]$ and $CR = [0.1 : 0.1 : 0.9]$
dmss-DE-pap	$N_{max} = 16 \cdot D, N_{min} = 10$, Operator pool size $H = 5$ F,CR, $\mu_F = 0.5, \mu_{CR} = 0.5$ $\alpha = 0.6, \beta = 0.1$,$P_p = 0.1$

With the help of updated quality values, weight values associated with underlying strategies are updated. In the next subsection, details about the proposed perturbation-based weight update mechanism are given.

Perturbation-Based Weight Update Mechanism

The weight value of the selected mutation strategy is updated using the quality values. If the updated quality value is $q_{k,t+1}$, and the quality value before applying the quality-update mechanism is $q_{k,t}$. These values are first compared, and if the updated quality value $q_{k,t+1}$ is better than the quality value $q_{k,t}$, the weight value of the selected mutation strategy is increased with the help of quality control parameter β, while keeping the weight values of other mutation strategies in the pool unchanged. Similarly, if the converse is true, that is, $q_{k,t+1}$ is less than or equal to $q_{k,t}$, the weight value associated with the selected mutation strategy is decreased, while keeping the weight values of other strategies unchanged. However, if a particular strategy is able to accumulate positive rewards in the early phase of the optimization procedure, its quality value, hence the weight value will be comparatively larger than the quality and weight values of other available mutation strategies. This will favor the strategy performing better at an early stage of the optimization procedure over other available mutation strategies in the pool, and the diversity of the pool will not be exploited optimally. Keeping this into consideration, a perturbation parameter P_p is introduced in the weight mechanism that will reinitialize the weight values of all available mutation strategies with a probability P_p. The implementation details of the proposed perturbation-based weight update mechanism are given in the Algorithm 5.

Algorithm 5 Perturbation-based Weight Update Mechanism

Require: selected mutation strategy $h_{k,t}$, quality value $q_{k,t}$, quality control parameter β, weight value $w_{k,t}$, Weight Vector W, Perturbation parameter P_p, iteration counter t.
Ensure: Weight value $w_{k,t+1}$, Weight vector W
at iteration t, if selected strategy is $h_{k,t}$,
if $rand() < P_p$ **then**
    $W \leftarrow rand(W)$ // assign random weight values to W.
else
    if $q_{k,t+1} > q_{k,t}$ **then**
        $w_{k,t+1} \leftarrow (1+\beta) * w_{k,t}$
    else
        $w_{k,t+1} \leftarrow (1-\beta) * w_{k,t}$
    end if
end if

The proposed perturbation-based weight update mechanism ensures that no particular mutation strategy in the strategy pool becomes greedy and dominates the optimization procedure while depleting the participation of other available mutation strategies in the pool. The proposed dmss-DE-pap algorithm utilizes two types of control parameters, one type is associated with the perturbation adaptive pursuit strategy while the other set is associated with the DE algorithm. In the next section, a discussion of associated parameters with the proposed dmss-DE-pap algorithm is given.

Associated Parameters

The proposed dmss-DE-pap algorithm utilizes two distinct sets of control parameters. One set of parameters contains the control parameters associated with the DE algorithm that are population size *N*, scaling factor *F*, and crossover rate *CR*. The other set of parameters contains the reward control parameter α, and quality control parameter β. In the proposed dmss-DE-pap algorithm, the control parameters associated with the DE algorithm are self-adaptive and majorly inspired by the LSHADE algorithm [31]. The population size is adjusted using the linear population size reduction (LPSR) mechanism as proposed in the LSHADE algorithm, and the control parameters are adjusted using the success-history-based parameter adaption scheme [32]. The implementation details of the success-history-based parameter adaption technique are given in algorithm 6. The self-adaptive control parameter setting improves the performance of the DE algorithm and reduces the burden of the computationally intensive task of manually tuning the control parameters.

Table 2 p-values at significance level 0.05 and effect size metric Cohens-d

CEC 14	30D			50D		
Algorithms	p-value	Cohens-d	h	p-value	Cohens-d	h
ABC	0.00176	0.7342	1	0.00085	0.8081	1
PSO	0.00072	0.8171	1	0.00052	0.8333	1
DE	0.00020	0.8389	1	0.00012	0.8771	1
CMAES	0.00274	0.6963	1	0.00384	0.5495	1
LSHADE	0.00979	0.5280	1	0.0560	0.1919	0
iLSHADE	0.00399	0.5661	1	0.00728	0.4319	1
jSO	0.05944	0.2022	0	0.00166	0.7311	1
SaDE	0.00190	0.7236	1	0.00613	0.5449	1
CoDE	0.00333	0.5719	1	0.00138	0.7417	1
EPSDE	0.00169	0.7316	1	0.00797	0.5099	1

Algorithm 6 Success-history-based-parameter adaption scheme [32]

Require: $M_F \leftarrow [0.5, \ldots, 0.5]$, $M_{CR} \leftarrow [0.5, \ldots, 0.5]$, s.t. $|M_F| = |M_{CR}| = 5$, $S_F \leftarrow \phi$, and $S_{CR} \leftarrow \phi$,
Ensure: Scalar Factor F, Crossover rate CR

select index $id \leftarrow rand\{1, K\}$
Generate $F_{id} \leftarrow randc_i(M_F(id), 0.1)$
// $randc_i(\mu, \sigma)$ Cauchy distribution with mean μ and variance σ
Generate $CR_{id} \leftarrow randn_i(M_{CR}(id), 0.1)$
// $randn_i(\mu, \sigma)$ Normal distribution with mean μ and variance σ
Store F_{id} and CR_{id} associated with successful individuals in S_F and S_{CR} i.e.
$S_F \leftarrow S_F \cup F_{id}$, $S_{CR} \leftarrow S_{CR} \cup CR_{id}$,
if $S_F, S_{CR} \neq \phi$ **then**
    $M_F \leftarrow Meanw_L(S_F)$, and
$M_{CR} \leftarrow Meanw_A(M_{CR})$
// $Meanw_L$ and $Meanw_A$ are Lehmer mean and arithmetic mean, respectively.
(see SHADE Algorithm [32])
else
    $M_F \leftarrow M_F$, and $M_{CR} \leftarrow M_{CR}$
end if
return $F \leftarrow M_F(id)$ and $CR \leftarrow M_{CR}(id)$

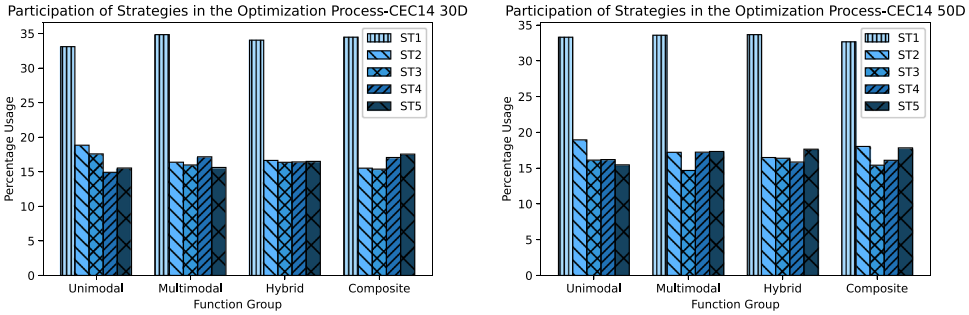
Another set of control parameters associated with perturbation-based adaptive pursuit strategy contains reward control parameter α and quality control parameter β. The values of α and β lies in the range (0, 1). In this research, the value of α is taken to be 0.6, and the value of β is taken to 0.1. A higher value of α near to or equal to 1 makes the population more greedy which could affect the exploration capabilities of the algorithm and can lead to premature convergence, on the other hand, a small value of α can affect the exploitation capabilities. The parameter β helps in scaling the quality vector. A value of β near 1 can be considered an aggressive quality update, while a value near 0 can be considered a pessimistic quality update. The value of the control parameters α and β are obtained empirically in this research by applying grid search on 10 equally spaced values of α and β in the range (0,1). For assessing the optimization capabilities of the proposed dmss-DE-pap algorithm, experiments on 30D and 50D CEC 2014 benchmark test suits are conducted, and experimental results are compared with the other state-of-the-art evolutionary algorithms and some recent DE variants. In the next section, experimental results obtained by the proposed dmss-DE-pap algorithm are reported in detail.

Experimental Results

The performance of the proposed dmss-DE-pap algorithm is tested on the CEC 2014 benchmark test suite [33]. The CEC 2014 benchmark test suite contains four different classes of optimization problems including unimodal problems, multi-modal problems, hybrid problems, and composite problems. In addition, these problems are highly anti-symmetric, rotated, and irregular [33]. Moreover, while solving these problems the information about their optimum solution is not used, which means these problems are treated as completely black-box problems. For our experiments, the CEC 2014 benchmark problems are labeled as $f_{a_1} - f_{a_{30}}$ and separated into four different groups: $f_{a_1} - f_{a_3}$ are in the unimodal functions group, $f_{a_4} - f_{a_{16}}$ are in the multi-modal function group, $f_{a_{17}} - f_{a_{22}}$ are in the hybrid function group, and $f_{a_{23}} - f_{a_{30}}$ are in the composite function group. To validate

Table 3 Average Ranks in the Friedman Test

Algorithms	CEC14		Avg	Final
	30D	50D	Ranks	Ranks
ABC	7.66	8.46	7.69	8
PSO	7.93	8.70	7.98	9
DE	10.03	10.33	10.03	11
CMAES	7.11	7.55	7.11	7
LSHADE	4	4.2	4.05	4
iLSHADE	3.75	3.5	3.74	3
jSO	3.71	4.16	3.74	2
SaDE	6.81	5.61	6.78	6
CoDE	8.06	6.73	8.01	10
EPSDE	4.96	4.15	4.91	5
dmss-DE-pap	1.93	2.58	1.91	**1**



(a) 30D CEC2014 Functions (b) 50D CEC2014 Functions

Fig. 2 Mean percent use of strategy pool in the optimization of 30D and 50D CEC 2014 Functions

the proposed dmss-DE-pap performance, four standard evolutionary algorithms, ABC [34], PSO [35], DE [8], CMA-ES [36], and three control parameter adaptive DE variants, LSHADE [31], i-LSHADE [37], and jSO [38], and three DE variants with multiple mutation strategies, SaDE [16], CoDE [18], and EPSDE [17] are considered for comparison. The control parameter settings are taken as mentioned in their original papers and given in Table 1.

As per the experimentation guidelines of CEC 2014 [33], 51 independent runs are carried out on each benchmark test function with an upper limit of $10000 \cdot D$ function evaluations, where D is the dimension of the search space. Test results for $30D$, and $50D$ are reported in this research, and fitness errors that are smaller than "eps" ($eps = 10^{-08}$) are considered to be zero. All the experiments are performed on a PC with Intel(R) Core(TM) i7-9750 H CPU @ 2.60GHz on the Windows 11 Operating System. For comparing the results of the proposed dmss-DE-pap algorithm with other considered algorithms, non-parametric tests are conducted and discussed in the next subsection.

Non-Parametric Tests

For comparing the proposed dmss-DE-pap with other underlying algorithms, the Wilcoxon signed-rank test is employed with the 5% significance level. The null hypothesis H_0 : "there is no significant difference between dmss-DE-pap and other compared algorithm" is tested based on the median values of mean fitness on 30D, and 50D test functions. The corresponding p-values at the significance level 5% and effect size metrics Cohens-d are reported in Table 2. The value '$h = 0$' indicates the failure to reject the null hypothesis H_o, and the value '$h = 1$' indicates the rejection of the null hypothesis H_o at 5% level of significance. The effect size metric Cohens-d provides the magnitude of the difference between the means of two populations. This allows to study the impact of different treatments over the underlying experiments. The cohens-d values can be obtained in four different scenarios: (1) simple group design, (2) two-group design, (3) single group two-repeated measures, (4) designs with baseline to compare. A detailed study concerning computing methodologies of cohens-d values can be found in [39]. In this research, a two group design is considered for which cohens-d values are calculated using following Eq. (7).

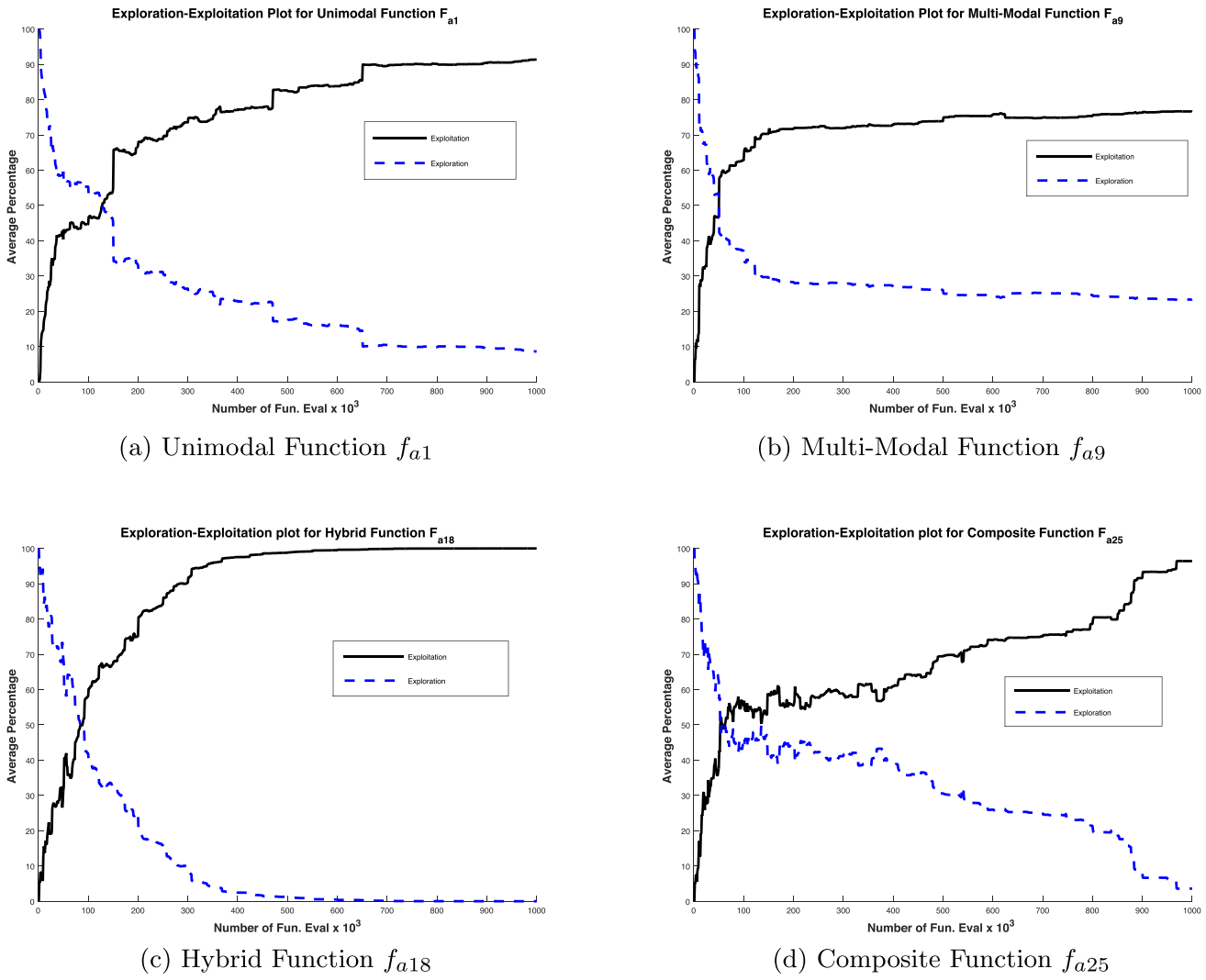
$$Cohens - d = \frac{M2 - M1}{S_p} \tag{7}$$

Where, $M1$, and $M2$ are mean values of group 1 and group 2, respectively. And, S_p is called pooled standard deviation and calculated using the Eq. (8).

$$S_p = \sqrt{\frac{(n_1 - 1) \cdot S_1^2 + (n_2 - 1) \cdot S_2^2}{(n_1 + n_2) - 2}} \tag{8}$$

Where, S_1 and S_2 are the standard deviation of the group 1 and group 2, respectively. And, n_1 and n_2 are a sample size of group 1 and group 2, respectively. The value of Cohens-d is subjective and depends on the experimental specifications. Values near 0.2 are considered as 'merely statistical', values between 0.2–0.5 are considered as 'subtle', and values above 0.5 are considered 'obvious' [39]. The proposed dmss-DE-pap is considered as group 1 while other compared algorithms are treated as group 2. The cohens-d values obtained by the proposed dmss-DE-pap algorithm are mostly greater than 0.2, except for the jSO algorithm in 30D problems and the LSHADE algorithm in 50D problems. This implies that the performance of the proposed dmss-DE-pap is superior compared to other considered algorithms.

To assess the winning performance of the dmss-DE-pap algorithm, the Friedman rank test is employed. The average rank and final rank obtained using the Friedman test are given in Table 3. It can be seen from Table 3, that the dmss-DE-pap algorithm outperforms all other underlying algorithms and secures the best rank. Additionally, for reporting the comprehensive performance of the proposed dmss-DE-pap algorithm from the perspective of the accuracy of the optimal solution, the **mean**, and **std** (standard deviation) of error for the total 51 runs are calculated and reported in Tables 5, and 6, respectively. The Rank/Stats below the test metrics **mean**, and **std** in the Tables 5 and 6, denote the



(a) Unimodal Function f_{a1}

(b) Multi-Modal Function f_{a9}

(c) Hybrid Function f_{a18}

(d) Composite Function f_{a25}

Fig. 3 Exploration-Exploitation Plots for Unimodal, Multi-Modal, Hybrid and Composite Functions

Table 4 Time complexity of dmss-DE-pap on 30D and 50D CEC14 benchmark

	$T0$	$T1$	$\hat{T}2$	$(\hat{T}2 - T1)/T0$
$D = 30$	0.32	3.26	4.16	2.81
$D = 50$	0.32	3.40	5.18	5.56

Friedman rank obtained by the considered algorithms on individual benchmark problems.

The proposed dmss-DE-pap algorithm utilizes a pool of 5 mutation strategies, however, it is important to analyze the participation of each mutation strategy in the optimization procedure. The next subsection discusses strategy pool analysis.

Strategy Pool Analysis

In the strategy pool H, 5 different mutation strategies are denoted as ST1-ST5, where ST1 is 'DE/current-to-pbest/1', ST2 is 'DE/rand/2', ST3 is 'DE/best/2', ST4 is 'DE/rand/1' and ST5 is 'DE/best/1'. To assess the contribution of each strategy in the search procedure, the mean percent use of all the strategies is reported in Fig. 2.

Since the CEC14 benchmark test suite is divided into four groups as unimodal functions group ($f_{a_1} - f_{a_3}$), multi-modal functions group ($f_{a_4} - f_{a_{16}}$), hybrid functions group ($f_{a_{17}} - f_{a_{22}}$), and composite functions group ($f_{a_{23}} - f_{a_{30}}$), the mean percent is calculated as the ratio of the number of invocation of a particular strategy to the total number of invocations of all the strategies in 51 runs. It can be observed that all the strategies in the pool are participating in the

Table 5 Performance of dmss-DE-pap on 30D CEC14 benchmark problems

Function	Metrics	ABC	PSO	DE	CMAES	LSHADE	iLSHADE	jSO	SaDE	CoDE	EPSDE	dmss-DE-pap
f_{a_1}	Mean	1168897	673206	125542754	1388214	0.00	0.00	0.00	34323459	177655987	3622625.7824	0.00
	std	389327	252991	1582181	647313	0.00	0.00	0.00	8704418	33580505	1110630	0.00
	Rank/Stats	6	5	10	7	2.5	2.5	2.5	9	11	8	2.5
f_{a_2}	Mean	102027.3189	24864.6837	2929.0943	25368.8104	0.00	0.00	0.00	115.6648	361005.7881	2013.2474	0.00
	std	103090.7139	9186.3291	2921.7263	15562.2862	0.0000	0.0000	0.0000	296.6460	82285.3629	2585.5101	0.00
	Rank/Stats	10	8	7	9	2.5	2.5	2.5	5	11	6	2.5
f_{a_3}	Mean	96781.7206	1888.3283	2400.7337	25328.0854	0.00	0.00	0.00	0.0385	38.3295	493.9442	0.00
	std	16215.7535	1167.8096	98195.9300	7381.7969	0.00	0.00	0.00	0.0165	9.4302	377.3316	0.00
	Rank/Stats	11	8	9	10	2.5	2.5	2.5	5	6	7	2.5
f_{a_4}	Mean	65.6047	83.3716	67.3304	**14.6980**	58.7064	58.5608	31.2856	85.9266	37.5361	27.3944	58.5192
	std	16.8246	24.4072	80.4583	0.9681	1.0119	0.0279	0.3011	2.9396	11.0132	5.6685	**0.00039**
	Rank/Stats	8	10	9	1	7	6	4	11	5	3	2
f_{a_5}	Mean	20.9264	20.7092	20.9011	20.3780	20.7617	20.9095	20.1993	20.8874	20.5615	20.4725	**19.1078**
	std	0.0731	0.1014	0.0468	0.0556	0.0211	0.0271	0.0878	0.0547	0.0482	0.0411	**0.0017**
	Rank/Stats	11	6	9	3	7	10	2	8	5	4	1
f_{a_6}	Mean	26.6472	32.5120	36.6145	**0.00002**	2.0856	2.9404	0.1191	18.0292	29.9198	16.8549	0.0095
	std	1.3645	2.4513	1.1100	**0.000004**	1.1974	1.1653	0.3891	9.8173	1.0102	1.4075	0.0326
	Rank/Stats	8	10	11	7	2	3	4	6	9	5	1
f_{a_7}	Mean	0.0092	0.1014	27.1047	0.0000	0.0001	0.0000	0.0000	0.0000	0.1046	0.0000	0.0000
	std	0.0066	0.0269	2.6779	0.0000	0.0035	0.0000	0.0000	0.0000	0.0277	0.0000	0.0000
	Rank/Stats	8	9	11	3.5	7	3.5	3.5	3.5	10	3.5	3.5
f_{a_8}	Mean	126.9828	75.9346	207.3757	158.5294	2.0544	0.0001	0.0004	0.9950	0.0047	0.0000	**00.00**
	std	11.2613	12.1327	12.3868	9.9794	0.6871	0.0000	0.0003	0.9543	0.0025	0.0000	**00.00**
	Rank/Stats	9	8	11	10	7	3	4	6	5	1.5	1.5
f_{a_9}	Mean	139.8936	110.1921	253.8467	158.2366	13.9078	12.2128	7.2312	154.7787	151.3456	65.6953	**6.4307**
	std	16.9042	24.2378	14.2111	9.4734	2.4240	2.3019	1.5592	9.6384	10.5750	8.6894	**1.0128**
	Rank/Stats	7	6	11	10	4	3	2	9	8	5	1
$f_{a_{10}}$	Mean	4698.9922	3374.6403	5963.2272	7185.4471	0.0606	0.0093	0.0019	41.6054	1481.6673	0.9133	**0.000551**
	std	305.7552	666.9198	194.1397	256.5499	0.0874	0.0113	0.0102	65.2327	150.6053	0.9784	**0.0013**
	Rank/Stats	9	8	10	11	3	2	1	5	7	4	6
$f_{a_{11}}$	Mean	4925.8594	4012.8754	7118.9788	7094.5264	264.4954	245.0340	**166.1397**	6754.3063	6467.4608	4246.4397	178.0536
	std	293.2717	545.5251	265.9878	219.6127	97.2161	72.1914	59.7051	238.3266	258.2134	409.6193	**54.5111**
	Rank/Stats	7	5	11	10	1	2	4	9	8	6	3
$f_{a_{12}}$	Mean	1.6806	1.0531	2.6333	**00.00**	0.2061	0.2357	0.2211	2.4434	1.4302	1.0602	0.2773
	std	0.2621	0.6139	0.3506	**00.00**	0.0319	0.0311	0.0412	0.2120	0.1457	0.1289	0.1031
	Rank/Stats	9	6	11	1	2	4	3	10	8	7	5
$f_{a_{13}}$	Mean	0.3619	0.5788	0.9710	0.2383	0.1133	0.1161	0.2040	0.3304	0.5090	0.3436	**0.0148**

Table 5 (continued)

Function	Metrics	ABC	PSO	DE	CMAES	LSHADE	iLSHADE	jSO	SaDE	CoDE	EPSDE	dmss-DE-pap
	std	0.0422	0.1355	0.1029	0.0373	0.0135	0.0135	0.0312	0.0336	0.0671	0.0446	**0.0026**
	Rank/Stats	8	10	11	5	2	3	4	6	9	7	1
$f_{a_{14}}$	Mean	0.2491	0.3077	0.6100	0.4162	0.0203	0.0020	0.0021	0.2862	0.2834	0.2709	**0.0000196**
	std	0.0226	0.0810	0.1931	0.0595	0.0253	0.0240	0.0239	0.0234	0.0336	0.0422	**0.000023**
	Rank/Stats	5	9	11	10	4	2	3	8	7	6	1
$f_{a_{15}}$	Mean	14.7075	22.7339	77.6716	14.0909	2.7164	2.3806	2.8917	14.9035	18.6018	9.9542	**0.2361**
	std	1.4596	3.5551	38.5173	1.0259	0.8111	0.2937	0.3866	1.1156	1.3621	1.2923	**0.2640**
	Rank/Stats	7	10	11	6	3	2	4	8	9	5	1
$f_{a_{16}}$	Mean	12.8753	12.3136	13.1742	13.0425	9.6370	**9.0046**	10.3822	12.9569	12.9918	11.7633	10.0735
	std	**0.1691**	0.4048	0.1833	0.3029	0.4541	0.4288	0.3454	0.1584	0.2107	0.2158	0.3991
	Rank/Stats	7	6	11	10	2	1	4	8	9	5	3
$f_{a_{17}}$	Mean	664220.2387	52330.8831	73665.5689	145447.5760	311.8816	270.7522	233.5549	157025.2361	3938082.3394	1497952.4534	**194.04185**
	std	162512.5504	8250.8831	2420423.1983	75270.9849	915.1032	102.5760	101.0881	84164.4918	881294.4146	760195.9547	**99.0017**
	Rank/Stats	9	5	6	7	4	3	2	8	11	10	1
$f_{a_{18}}$	Mean	322976.8138	12322.6385	48363.4231	155152.3660	6.2068	7.3190	7.7831	2610991.9426	79365146.2362	126281.5015	**5.3952**
	std	20356.2235	1515.3403	13460.3711	94617.6379	2.5867	2.9330	2.8989	1629776.2498	25604899.7624	78978.4359	**2.5771**
	Rank/Stats	9	5	6	8	1	3	4	10	11	7	2
$f_{a_{19}}$	Mean	20.1091	19.6606	44.6085	28.9247	7.5610	7.6120	9.1517	15.7585	19.0424	13.1393	**3.5702**
	std	2.0563	14.6210	48.3448	3.4903	2.0087	1.6962	2.0937	1.7698	0.7878	1.2911	**1.1721**
	Rank/Stats	9	8	11	10	2	3	4	6	7	5	1
$f_{a_{20}}$	Mean	12973.6539	856.9803	5867.2463	38395.9982	4.7668	3.2319	3.3421	264.8438	14132.1714	2845.0406	**2.2115**
	std	4506.3813	407.9382	181.3690	22970.1954	2.8235	1.0828	1.4417	75.0821	4812.4547	1222.7287	**1.0163**
	Rank/Stats	9	6	8	11	4	2	3	5	10	7	1
$f_{a_{21}}$	Mean	61753.3329	24177.5866	75932.2937	27955.9131	179.7922	160.7902	107.8432	6995.8956	1073094.9953	32322.9976	**102.1273**
	std	22175.4906	9351.9790	17861.8336	12343.8432	79.5114	95.1899	74.2052	2600.7399	401347.5963	17506.5561	**73.0134**
	Rank/Stats	9	6	10	7	2	3	4	5	11	8	1
$f_{a_{22}}$	Mean	196.7926	400.5809	792.2238	931.5592	**26.9921**	27.1423	28.3109	440.5981	844.8080	116.8092	31.2729
	std	75.2578	206.6802	111.7555	214.3385	**1.1888**	1.3814	1.8396	108.6903	157.1322	43.8416	4.1887
	Rank/Stats	6	7	9	11	1	2	3	8	10	5	4
$f_{a_{23}}$	Mean	330.9728	335.9160	361.3401	334.1911	345.8287	335.5828	**331.3936**	335.5795	330.1427	330.1422	339.7985
	std	0.2761	0.4110	3.1679	6.8400	0.0000	0.0000	0.0001	0.0000	0.0015	0.0023	**00.00**
	Rank/Stats	4	9	11	6	10	8	5	7	2	1	3
$f_{a_{24}}$	Mean	201.3411	201.4821	202.4973	201.0409	201.1266	201.0467	201.7782	200.9526	201.2568	200.9871	**200.0699**
	std	0.0880	0.2101	0.4700	24.3866	0.0621	0.0473	0.0704	0.0541	0.0886	0.0761	**0.0380**
	Rank/Stats	8	9	11	4	6	5	10	2	7	3	1
$f_{a_{25}}$	Mean	206.0698	230.7797	240.7502	206.8729	209.5240	201.9778	202.3442	210.8443	228.0184	201.9278	**191.5028**
	std	1.2100	16.4564	6.2349	2.1778	1.3302	1.3880	1.6164	2.1842	4.6821	0.6616	**0.4081**

Table 5 (continued)

Function	Metrics	ABC	PSO	DE	CMAES	LSHADE	iLSHADE	jSO	SaDE	CoDE	EPSDE	dmss-DE-pap
	Rank/Stats	5	10	11	6	7	3	4	8	9	2	1
f_{a26}	Mean	107.3028	104.4091	107.6969	102.2497	102.1773	102.3097	102.2067	100.3330	100.5010	100.3208	**100.0890**
	std	0.0375	0.1061	0.0700	0.0392	0.0309	0.0132	0.0154	0.0451	0.0587	0.0422	**0.00138**
	Rank/Stats	10	9	11	7	5	8	6	3	4	2	1
f_{a27}	Mean	559.5266	952.8252	1142.6144	779.0900	360.6243	326.1744	302.0856	395.2821	1106.9915	550.7734	**300.0021**
	std	163.1910	218.5664	53.5632	471.5412	31.6390	19.1741	3.6940	49.0600	26.3717	128.6288	**2.6711**
	Rank/Stats	7	9	11	8	4	3	2	5	10	6	1
f_{a28}	Mean	421.4231	1247.3186	823.9053	419.5041	418.6258	421.5217	**415.5217**	421.3139	534.3331	395.3532	415.9995
	std	9.3339	286.4032	43.5664	8.7465	2.1008	3.2212	5.8798	5.9007	32.8959	3.8546	**3.1417**
	Rank/Stats	6	11	10	4	8	7	3	5	9	2	1
f_{a29}	Mean	**228.4343**	6344101.4827	6618804.8067	**227.6762**	597.3802	430.1388	428.3178	507087.5043	226.5622	213.7070	429.6009
	std	**2.1985**	1342652.9414	101801.2401	2.5833	104.4827	151.4143	5.2497	2454306.2059	2.5807	1.0084	5.2131
	Rank/Stats	3	10	11	2	4	7	8	9	6	5	1
f_{a30}	Mean	612.5759	2413.4096	2761.2874	1714.8557	485.8139	**422.6412**	435.4469	710.5427	1610.4813	427.9022	440.6659
	std	87.6822	552.5512	561.3067	213.7392	**45.2903**	38.9796	52.8254	175.7227	218.3453	86.4344	420.0818
	Rank/Stats	6	10	11	9	5	2	4	7	8	3	1

optimization procedure. However, strategy ST1 is the most prominent strategy among others as it has maximum contribution in the optimization process. It is also important to analyze the exploration-exploitation capabilities of the proposed dmss-DE-pap algorithm for the better understanding of its optimization capabilities. In the next section, an analysis of the exploration-exploitation capabilities of the proposed dmss-DE-pap algorithm is discussed.

Exploration-Exploitation Analysis

Apart from managing the diversity of the pool, the proposed dmss-DE-pap algorithm also maintains a balance between exploration and exploitation of the search procedure. For effectively assessing the exploration-exploitation trade-off, a dimension-wise diversity-based analysis is adopted [40]. The percentage of exploitation and exploration is calculated using the equation (9) given below:

$$\begin{aligned} Exploitation\% &= \left(\frac{|Div - Div_{max}|}{Div_{max}}\right) \times 100 \\ Exploration\% &= \left(\frac{Div}{Div_{max}}\right) \times 100 \end{aligned} \tag{9}$$

Here, the diversity Div is calculated using the equation (10) given below:

$$\begin{aligned} Div &= \frac{1}{d}\sum_{j=1}^{d} Div_j \\ Div_j &= \frac{1}{N}\sum_{i=1}^{N} \left| med_j(X) - x_{i,j} \right| \end{aligned} \tag{10}$$

Div_{max} is the maximum diversity recorded during the optimization process. Here, $med_j(X)$ denotes the median of the j^{th} dimension in the population. To assess the exploration-exploitation capability of the dmss-DE-pap algorithm, population diversity analysis is performed on unimodal (f_{a1}), multi-modal (f_{a9}), hybrid (f_{a18}), and composite (f_{a25}) functions of 30D CEC14 benchmark problems, and exploration-exploitation plots are illustrated in Fig. 3.

The proposed dmss-DE-pap algorithm depicts a fine balance between exploration and exploitation during its optimization procedure, hence it can be considered as a robust optimizer for solving complex optimization problems. In the next subsection, the algorithmic time complexity of the proposed dmss-DE-pap algorithm is reported.

Algorithmic Time Complexity

The algorithmic time complexity of the proposed dmss-DE-pap algorithm is evaluated using the criteria given by the CEC14 benchmark test problems [33]. Table 4 shows the

Table 6 Performance of dmss-DE-pap on 50D CEC14 benchmark problems

Function	Metrics	ABC	PSO	DE	CMAES	LSHADE	iLSHADE	jSO	SaDE	CoDE	EPSDE	dmss-DE-pap
f_{a_1}	Mean	186992223	5107214	991878678	507135346298	36.1047	6228.2958	13.8309	34847815	178928223	3786515	**0.7795**
	std	33849983	1482719	159455060	17175962	29.6609	**8.7721**	11.9793	10032890	33505735	1229410	13.1505
	Rank/ stats	9	6	10	11	3	4	2	7	8	5	1
f_{a_2}	Mean	157902.9099	234984.1995	38397737.1174	43746.9728	0.1461	0.0000	0.0000	177.2816	347102.4377	2461.3145	0.0000
	std	56791.5415	51859.6393	4123317.6210	35290.2302	0.3021	0.0000	0.0000	729.5583	100559.8986	2572.1338	0.0000
	Rank/ stats	8	9	11	7	4	2	2	5	10	6	2
f_{a_3}	Mean	231355.7385	5283.1528	12918.9355	51250.9498	0.0000	0.0000	0.0000	0.0372	45.1075	598.8065	0.0000
	std	30271.7184	2785.1342	7807.8937	94666.9485	0.0000	0.0000	0.0000	0.0121	12.6003	495.0484	0.0000
	Rank/ stats	11	8	9	10	2.5	2.5	2.5	5	6	7	2.5
f_{a_4}	Mean	184.3219	165.0698	5894.2871	**33.4045**	73.1254	73.0454	91.7000	85.4815	36.1012	30.6541	73.1000
	std	16.0398	48.1142	639.8726	**4.0093**	46.4629	51.1162	44.8098	0.1139	8.5971	14.8281	48.2110
	Rank/ stats	10	9	11	2	6	4	8	7	3	1	5
f_{a_5}	Mean	21.1415	21.0490	21.1642	21.1710	20.5425	20.1065	**20.0330**	20.8873	20.5699	20.4545	20.0423
	std	0.0301	0.0906	0.0358	0.0330	0.0385	0.0425	**0.0247**	0.0535	0.0415	0.0364	0.0409
	Rank/ stats	9	8	10	11	5	3	1	7	6	4	2
f_{a_6}	Mean	55.2286	62.2523	70.0376	**0.0010**	9.4125	0.4971	1.5700	18.1154	29.7123	17.7438	0.5240
	std	2.4553	4.2010	1.6400	**0.00025**	1.9851	0.7331	0.9630	9.2602	0.9783	1.5174	0.0697
	Rank/ stats	9	10	11	1	5	2	4	7	8	6	3
f_{a_7}	Mean	0.2181	0.4349	355.5692	0.0008	0.00	0.00	0.00	0.00	0.1092	0.00	**0.00**
	std	0.0327	0.0815	26.8588	0.00	0.00	0.0073	0.00	0.00	0.0302	0.00	**0.00**
	Rank/ stats	9	10	11	7	3.5	3.5	3.5	3.5	8	3.5	3.5
f_{a_8}	Mean	383.6152	193.2729	481.1875	223.2433	14.2667	0.0001	0.0000	0.6633	0.0043	0.0195	**00.00**
	std	26.0401	15.5619	14.7044	14.2452	12.6570	0.00	0.00	0.8513	0.0016	0.1379	**0.00**
	Rank/ stats	10	8	11	9	7	3	1.5	6	4	5	1.5
f_{a_9}	Mean	396.3941	326.3418	583.0587	274.4910	**10.7553**	11.1639	13.6724	154.0006	154.7836	66.9561	19.4018
	std	33.1120	33.0177	19.9188	106.5880	**2.0513**	2.2700	3.1971	9.0858	9.9886	8.3930	2.5209
	Rank/ stats	10	9	11	8	1	2	3	6	7	5	4
$f_{a_{10}}$	Mean	10124.1008	6166.1133	11954.7699	13250.4730	6.5274	0.0501	14.9015	45.2914	1482.4858	0.7656	**0.04712**
	std	559.2415	620.3098	320.2191	349.9108	3.5435	0.0209	21.6012	74.2884	133.3584	0.8223	**0.0266**

Table 6 (continued)

Function	Metrics	ABC	PSO	DE	CMAES	LSHADE	iLSHADE	jSO	SaDE	CoDE	EPSDE	dmss-DE-pap
	Rank/stats	9	8	10	11	4	2	5	6	7	3	1
$f_{a_{11}}$	Mean	10774.9561	7149.5598	13753.8457	13265.5933	4650.7150	4240.1924	4720.8623	6685.3627	6393.4829	4144.1376	**4180.1501**
	std	401.7628	888.6226	335.0519	444.7733	366.1924	306.3243	319.1867	339.9379	275.3626	523.1222	**301.4393**
	Rank/stats	9	8	11	10	4	3	5	7	6	1	2
$f_{a_{12}}$	Mean	2.7061	1.8275	3.6195	3.7113	0.4881	0.4827	**0.4261**	2.3523	1.4671	1.0568	0.4628
	std	0.3071	0.5532	0.3445	0.3039	0.0642	0.0697	**0.0538**	0.2212	0.1675	0.1504	0.0665
	Rank/stats	9	7	10	11	4	3	1	8	6	5	2
$f_{a_{13}}$	Mean	0.4266	0.5229	0.5102	0.2358	0.1591	**0.1511**	0.2620	0.3366	0.5028	0.3448	0.2211
	std	0.0397	0.1030	0.2468	0.0373	0.0173	**0.0020**	0.0322	0.0407	0.0552	0.0401	0.0254
	Rank/stats	8	11	10	4	2	1	5	6	9	7	3
$f_{a_{14}}$	Mean	0.3267	0.3919	7.2084	0.4210	0.2637	0.2681	0.2803	0.2817	0.2830	0.2988	**0.2521**
	std	0.0247	0.1173	6.8640	0.0738	0.0190	0.0167	0.0249	0.0273	0.0319	0.0451	**0.0188**
	Rank/stats	8	9	11	10	2	3	4	5	6	7	1
$f_{a_{15}}$	Mean	42.3152	55.5548	1353.8417	26.7110	5.6931	7.4509	**5.6111**	15.3361	18.7787	9.6845	5.9702
	std	2.3194	9.6901	430.6032	3.6933	0.6341	1.2021	**0.5740**	0.9013	1.4059	1.0004	0.6344
	Rank/stats	9	10	11	8	2	4	1	6	7	5	3
$f_{a_{16}}$	Mean	22.0801	20.2396	22.3960	22.1708	17.2540	17.1049	**17.0091**	12.9431	13.0123	11.7850	17.1098
	std	0.2148	0.9219	0.1756	0.5160	0.4923	0.6231	**0.3746**	0.1925	0.2190	0.3174	0.5141
	Rank/stats	9	8	11	10	7	5	4	2	3	1	6
$f_{a_{17}}$	Mean	95100.7656	51476.4581	76790.9372	18745.1572	1515.7234	1413.1335	2804.0801	166170.3157	3896147.9114	1171515.8825	**995.1313**
	std	25875.0585	4581.6240	13955.4806	4408.0113	392.3532	388.6026	575.3220	79368.1540	831705.2111	595886.0309	**310.7341**
	Rank/stats	8	6	7	5	3	2	4	9	11	10	1
$f_{a_{18}}$	Mean	16884219.0450	4190.3699	9077091.2167	161227.9279	257.2820	230.3933	351.2441	3029619.0909	72994010.5328	142928.6767	**44.9165**
	std	64294.6590	2274.2989	1514538.6896	63544.1688	71.2401	85.3569	40.9777	1966668.2281	23687709.5843	87724.6676	**13.3327**
	Rank/stats	10	5	9	7	3	2	4	8	11	6	1
$f_{a_{19}}$	Mean	46.0120	33.3522	147.1763	24.4775	15.3983	**14.4010**	16.2321	15.5763	18.9826	12.9816	15.0809
	std	2.5447	12.5781	13.1000	1.2519	5.0007	5.2777	**2.3201**	2.1214	0.8431	1.4280	2.7995
	Rank/stats	10	9	11	8	4	2	6	5	7	1	3

Table 6 (continued)

Function	Metrics	ABC	PSO	DE	CMAES	LSHADE	iLSHADE	jSO	SaDE	CoDE	EPSDE	dmss-DE-pap
$f_{a_{20}}$	Mean	122403.5942	2463.2145	18836.6729	146522.6760	203.1641	17.2641	18.2531	257.1996	13896.7464	2916.6771	**13.2925**
	std	36198.4966	1211.3030	6650.5980	64886.5758	38.2544	5.2300	7.3119	67.6376	5498.2999	1345.0557	**3.3201**
	Rank/ stats	10	6	9	11	4	2	3	5	8	7	1
$f_{a_{21}}$	Mean	456503.9386	99113.4703	112239.8501	713320.1307	524.5587	541.3072	1460.3203	7237.9623	1209342.0006	34596.9860	**319.2516**
	std	152481.0086	79763.4067	27792.6246	290340.9671	207.3393	208.4878	337.1748	1822.1141	420303.3923	21028.9461	**137.9705**
	Rank/ stats	9	7	8	10	2	3	4	5	11	6	1
$f_{a_{22}}$	Mean	1179.0487	1175.8190	2037.0174	1674.8082	154.3545	49.8203	35.0104	433.4201	833.3047	113.9965	**32.1614**
	std	138.9263	495.4630	191.8967	216.3077	33.1042	28.6177	6.4041	112.4060	149.5611	45.6993	**4.6501**
	Rank/ stats	9	8	11	10	5	3	2	6	7	4	1
$f_{a_{23}}$	Mean	**338.8963**	346.8462	430.7476	341.6622	344.3961	343.0335	341.6622	335.5795	330.1428	330.1425	344.2980
	std	0.5690	2.2393	8.4671	3.1620	0.0018	0.0117	0.0315	0.0000	0.0013	0.0034	**0.00173**
	Rank/ stats	4	10	11	5.5	9	7	5.5	3	2	1	8
$f_{a_{24}}$	Mean	201.3227	229.6259	277.7168	225.0153	206.0502	203.2369	201.5390	200.9938	201.2616	200.9921	**200.0153**
	std	0.1240	85.0712	7.7682	74.5506	50.8902	**49.2567**	94.0932	0.1209	0.0669	0.0557	67.2892
	Rank/ stats	5	10	11	9	8	7	6	3	4	2	1
$f_{a_{25}}$	Mean	236.6349	281.3194	354.8999	211.0297	206.9077	205.8811	208.9325	210.9619	228.3030	201.8462	**205.5974**
	std	5.4533	34.1672	18.2235	2.6835	0.2721	0.2030	0.2499	2.3559	6.4252	0.5847	**0.2790**
	Rank/ stats	9	10	11	7	4	3	5	6	8	1	2
$f_{a_{26}}$	Mean	100.4068	100.4934	103.7011	100.2342	100.1552	100.2343	100.2766	100.3460	100.4784	100.3162	**100.1476**
	std	0.0443	0.0110	0.1701	0.0366	0.0176	**0.0157**	0.0233	0.0343	0.0541	0.0447	0.0209
	Rank/ stats	8	10	11	3	2	4	5	7	9	6	1
$f_{a_{27}}$	Mean	1346.3075	1959.2804	2003.0069	906.8431	300.0637	306.6927	302.0306	412.7478	1100.3125	569.7113	**300.0387**
	std	360.9844	718.5959	52.6073	871.9955	3.4517	9.2606	4.6801	72.4977	68.9990	138.7092	**2.1310**
	Rank/ stats	9	10	11	7	2	4	3	5	8	6	1
$f_{a_{28}}$	Mean	614.9289	2483.3948	2184.7686	**444.2976**	471.3473	474.3503	563.9406	420.3711	547.0072	395.5909	460.6351
	std	39.8840	521.9326	135.0883	12.5911	8.6113	12.5301	4.0566	3.6251	29.2400	4.1814	**4.2403**
	Rank/ stats	9	11	10	3	5	6	8	2	7	1	4
$f_{a_{29}}$	Mean	264.0130	603848.7281	716711.1465	**230.2599**	378.9859	378.8465	555.7136	249351.7772	226.3247	213.6560	368.6483
	std	8.2400	232363.6445	152319.4937	3.8992	6.4710	11.0177	9.3148	1736821.3899	2.6532	1.1526	**3.4811**

Table 6 (continued)

Function	Metrics	ABC	PSO	DE	CMAES	LSHADE	iLSHADE	jSO	SaDE	CoDE	EPSDE	dmss-DE-pap
	Rank/ stats	4	10	11	3	7	6	8	9	2	1	5
f_{a30}	Mean	**1790.5498**	22915.3044	21578.8620	2704.0511	2460.2248	2478.4517	3155.5700	703.6164	1585.5998	448.9277	1841.6201
	std	**203.7353**	7880.1780	4876.8370	278.2450	541.6218	543.1780	576.5434	183.1853	219.9451	78.6321	282.1690
	Rank/ stats	4	11	10	8	6	7	9	2	3	1	5

time complexity of the dmss-DE-pap algorithm on 30 and 50-dimensional benchmark problems.

$T0$ is the time taken (in seconds) to compute the test problem given as following [33]:

```
clc;
clear all;
tic
for i=1:1000000
x= 0.55 + [double]*i;
x=x + x; x=x/2; x=x*x; x=sqrt(x);
x=log(x); x=exp(x); x=x/(x+2);
end
toc
```

The time taken to compute 200000 function evaluations of f_{a18} problem is denoted by $T1$, and $\hat{T}2$ is the average time taken to compute 200000 function evaluations of benchmark function f_{a18} for five independent runs by dmss-DE-pap algorithm.

Discussion

The experimental findings concludes that the dmss-DE-pap algorithm demonstrates competitive performance when compared with other state-of-the-art algorithms. It can be noted from Fig. 2 that dmss-DE-pap effectively manages the underlying operator pool. All the strategies participate in the optimization process. Strategy ST1 in the operator pool reports the maximum mean percent usage and hence can be considered a better strategy compared to other strategies in the pool. Similarly, Fig. 3 depicts that the dmss-DE-pap algorithm effectively manages the exploration-exploitation trade-off. To compare the performance of the dmss-DE-pap algorithm with other underlying algorithms, Tables 5 and 6 reports the optimization performance of all underlying algorithms along with the dmss-DE-pap algorithm on CEC14 test suite. Better results are marked with bold entries in the Tables 5 and 6, and the corresponding Friedman ranks are given below the test metrics **mean** and **std**. The comparative assessment of the dmss-DE-pap algorithm on the CEC14 test suite is given in the following subsections.

Comparison on 30D CEC14 Benchmark Problems

For 30D CEC14 benchmark problems, the dmss-DE-pap algorithm secures the lowest rank on 17 out of 30 problems. The performance of the dmss-DE-pap algorithm on unimodal problems $f_{a1} - f_{a3}$ is better than all other underlying

algorithms except the performance is similar to LSHADE, i-LSHADE, and jSO algorithms. For multi-modal problems $f_{a4} - f_{a16}$, the dmss-DE-pap algorithm outperforms all other algorithms on f_{a5},f_{a6},f_{a9},f_{a13},f_{a14} and f_{a15}. On function f_{a4}, the dmss-DE-pap algorithm secures the second rank outperforming all other algorithms except CMA-ES. The performance of the dmss-DE-pap algorithm is similar to CMA-ES, i-LSHADE, jSO, SaDE, and EPSDE on function f_{a7}, and the performance of the dmss-DE-pap algorithm is similar to EPSDE on function f_{a8}. The dmss-DE-pap algorithm secures third rank on the functions f_{a11} and f_{a16}, fifth rank on the function f_{a12}, and sixth rank on the function f_{a10}. Similarly, on hybrid functions $f_{a17} - f_{a22}$, the dmss-DE-pap algorithm secures the best rank on f_{a17},f_{a19},f_{a20}, and f_{a21}, while it secures second rank on the function f_{a18} outperforming all other algorithms except LSHADE algorithm. The dmss-DE-pap algorithm secures the fourth rank on the function f_{a22}, behind LSHADE, i-LSHADE and jSO algorithm. On composite functions $f_{a23} - f_{a30}$, the dmss-DE-pap algorithm secures the best rank in all the functions except on the function f_{a23}, where the performance of the dmss-DE-pap algorithm is behind CoDE and EPSDE algorithm (See Table 5). The dmss-DE-pap received the best rank of 1.93 over all 30D functions.

Comparison on 50D CEC14 Benchmark Problems

For 50D CEC14 benchmark problems, the dmss-DE-pap algorithm secures the lowest rank on 11 out of 30 problems outperforming all other underlying algorithms. For unimodal function group $f_{a1} - f_{a3}$, the dmss-DE-pap algorithm outperforms all other algorithms on function f_{a1}. In contrast, the performance of the dmss-DE-pap algorithm is similar to i-LSHADE and jSO on f_{a2}, and the performance is similar to LSHADE, i-LSHADE, and jSO on function f_{a3}. Similarly, for multi-modal function group $f_{a4} - f_{a16}$, the dmss-DE-pap algorithm almost outperforms all other underlying algorithms. On function f_{a10},f_{a14}, the performance of dmss-DE-pap is similar to LSHADE, i-LSHADE, jSO, SaDE, and EPSDE, and on function f_{a7} and f_{a8}, its performance is similar to the jSO algorithm. The dmss-DE-pap algorithm secures second rank, on the functions f_{a5} and f_{a12} behind the jSO algorithm and secures second rank. On the function f_{a11}, it is behind the EPSDE algorithm. In a similar manner, for hybrid function group $f_{a17} - f_{a22}$, the dmss-DE-pap algorithm secures first rank on the functions f_{a17},f_{a19},f_{a20},f_{a21}, second rank on the function f_{a18}, and fourth rank on the function f_{a22}. For composite function group $f_{a23} - f_{a30}$, the dmss-DE-pap algorithm secures the first rank on the function except for function f_{a23} (refer Table 6) The experimental results confirm the competitive optimization capabilities of the proposed dmss-DE-pap algorithm.

Conclusion

The perturbation-based adaptive pursuit strategy is an effective strategy for operator selection from a given pool of operators. It manages the diversity of the pool better and enhances the robustness of the algorithm. In the proposed work, the mentioned strategy is employed on the pool of mutation operators of DE to exploit the various characteristics of multiple mutation operators in a single run of the optimization process. A success history-based parameter adaption scheme is employed to tune the control parameters F and CR. A linear population reduction mechanism is employed to check the computational cost of the algorithm. Experimental results of the proposed dmss-DE-pap are promising when compared with other state-of-the-art evolutionary and swarm intelligence-based algorithms. However, the proposed approach may fail, if all the mutation operators in the operator pool are not capable of handling the given optimization problem or suffer from poor performance. The optimal size of the operator pool and content of the operator pool in dmss-DE-pap is a matter of further research. More sophisticated advanced methods of reinforcement learning and machine learning can be incorporated. Apart from the adaptive pursuit strategies, upper confidence bound (UCB) methods can also be employed similarly and their effect on the performance of the current algorithmic framework can be studied in the future research.

Author Contributions Prathu Bajpai: Preparation of the original draft, and code implementation. Ogbonnaya Anicho: Review and Editing. Atulya K. Nagar: Conceptualization and supervision. Jagdish Chand Bansal: Conceptualization, proofreading, and supervision.

Data availability Not applicable.

Code Availability Code may be available on request for fair use/research.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no Conflict of interest or Conflict of interest.

Ethical Approval and Consent to Participate No human/animal participants are involved in this study.

Consent for Publication All Authors give their consent for publication of this manuscript.

References

1. Michalewicz Z, Dasgupta D, Le Riche RG, Schoenauer M. Evolutionary algorithms for constrained engineering problems. Comput Ind Eng. 1996;30(4):851–70.
2. Carvalho DR, Freitas AA. A hybrid decision tree/genetic algorithm method for data mining. Inf Sci. 2004;163(1–3):13–35.
3. Gobeyn S, Mouton AM, Cord AF, Kaim A, Volk M, Goethals PL. Evolutionary algorithms for species distribution modelling: a review in the context of machine learning. Ecol Model. 2019;392:179–95.
4. Simon, D. Evolutionary optimization algorithms. John Wiley & Sons, 2013.
5. Nocedal J, Wright S. Numerical optimization. Springer Science and Business Media. 2006.
6. Bäck T, Schwefel H-P. An overview of evolutionary algorithms for parameter optimization. Evol Comput. 1993;1(1):1–23.
7. Derrac J, García S, Hui S, Suganthan PN, Herrera F. Analyzing convergence performance of evolutionary algorithms: a statistical approach. Inf Sci. 2014;289:41–58.
8. Storn R, Price K. A simple and efficient heuristic for global optimization over continuous spaces. J Glob Optim. 1997;11:341–59.
9. Slowik A, Bialko M. Training of artificial neural networks using differential evolution algorithm. In: 2008 Conference on human system interactions, 2008;pp. 60–65. IEEE
10. Onwubolu G, Davendra D. Scheduling flow shops using differential evolution algorithm. Eur J Oper Res. 2006;171(2):674–92.
11. Jebaraj L, Venkatesan C, Soubache I, Rajan CCA. Application of differential evolution algorithm in static and dynamic economic or emission dispatch problem: a review. Renew Sustain Energy Rev. 2017;77:1206–20.
12. Das S, Suganthan PN. Differential evolution: a survey of the state-of-the-art. IEEE Trans Evol Comput. 2010;15(1):4–31.
13. Gämperle R, Müller SD, Koumoutsakos P. A parameter study for differential evolution. Adv intell Syst Fuzzy Syst Evol Comput. 2002;10(10):293–8.
14. Zaharie D. Control of population diversity and adaptation in differential evolution algorithms. Proc of MENDEL. 2003;9:41–6.
15. Das S, Konar A, Chakraborty UK. Two improved differential evolution schemes for faster global search. In: Proceedings of the 7th annual Conference on Genetic and Evolutionary Computation, 2005;pp. 991–998.
16. Qin AK, Huang VL, Suganthan PN. Differential evolution algorithm with strategy adaptation for global numerical optimization. IEEE Trans Evol Comput. 2008;13(2):398–417.
17. Mallipeddi R, Suganthan PN, Pan Q-K, Tasgetiren MF. Differential evolution algorithm with ensemble of parameters and mutation strategies. Appl Soft Comput. 2011;11(2):1679–96.
18. Wang Y, Cai Z, Zhang Q. Differential evolution with composite trial vector generation strategies and control parameters. IEEE Trans Evol Comput. 2011;15(1):55–66.
19. Wu G, Mallipeddi R, Suganthan PN, Wang R, Chen H. Differential evolution with multi-population based ensemble of mutation strategies. Inf Sci. 2016;329:329–45.
20. Wu G, Shen X, Li H, Chen H, Lin A, Suganthan PN. Ensemble of differential evolution variants. Inf Sci. 2018;423:172–86.
21. Qian W, Chai J, Xu Z, Zhang Z. Differential evolution algorithm with multiple mutation strategies based on roulette wheel selection. Appl Intell. 2018;48:3612–29.
22. Li Y, Wang S, Yang B. An improved differential evolution algorithm with dual mutation strategies collaboration. Expert Syst Appl. 2020;153: 113451.
23. Deng W, Shang S, Cai X, Zhao H, Song Y, Xu J. An improved differential evolution algorithm and its application in optimization problem. Soft Comput. 2021;25:5277–98.
24. Lampinen J, Zelinka I, *et al.* On stagnation of the differential evolution algorithm. In: Proceedings of MENDEL, 2000;vol. 6, pp. 76–83. Citeseer.
25. Thierens D. An adaptive pursuit strategy for allocating operator probabilities. In: Proceedings of the 7th annual conference on genetic and evolutionary computation, 2005;pp. 1539–1546.
26. Gong W, Fialho Á, Cai Z, Li H. Adaptive strategy selection in differential evolution for numerical optimization: an empirical study. Inf Sci. 2011;181(24):5364–86.
27. Goldberg DE. Probability matching, the magnitude of reinforcement, and classifier system bidding. Mach Learn. 1990;5:407–25.
28. Zhang S, Ren Z, Li C, Xuan J. A perturbation adaptive pursuit strategy based hyper-heuristic for multi-objective optimization problems. Swarm Evol Comput. 2020;54: 100647.
29. Thierens D. Adaptive strategies for operator allocation. Parameter setting in evolutionary algorithms, 77–90, 2007.
30. Mousavirad SJ, Rahnamayan S. Enhancing shade and l-shade algorithms using ordered mutation. In: 2020 IEEE symposium series on computational intelligence (SSCI), 2020;pp. 337–344. IEEE
31. Tanabe R, Fukunaga AS. Improving the search performance of shade using linear population size reduction. In: 2014 IEEE congress on evolutionary computation (CEC), 2014;pp. 1658–1665. IEEE
32. Tanabe R, Fukunaga A. Success-history based parameter adaptation for differential evolution. In: 2013 IEEE congress on evolutionary computation, 2013;pp. 71–78. IEEE
33. Liang JJ, Qu BY, Suganthan PN. Problem definitions and evaluation criteria for the cec 2014 special session and competition on single objective real-parameter numerical optimization. Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore 2013;**635**(2).
34. Karaboga D, Akay B. A comparative study of artificial bee colony algorithm, applied mathematics and computation. Appl Math Comput. 2009;214:108–32.
35. Kennedy J, Eberhart R. Particle swarm optimization. Proc IEEE Int Conf Neural Netw. 1995;4:1942–8.
36. Hansen N, Ostermeier A. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In: Proceedings of IEEE international conference on evolutionary computation, 1996;pp. 312–317. IEEE
37. Brest J, Maučec MS, Bošković B. il-shade: Improved l-shade algorithm for single objective real-parameter optimization. In: 2016 IEEE congress on evolutionary computation (CEC), 2016;pp. 1188–1195. IEEE
38. Brest J, Maučec MS, Bošković B. Single objective real-parameter optimization: Algorithm jso. In: 2017 IEEE congress on evolutionary computation (CEC), 2017;pp. 1311–1318. IEEE
39. Goulet-Pelletier J-C, Cousineau D. A review of effect sizes and their confidence intervals, part I: the cohen'sd family. Quant Methods Psychol. 2018;14(4):242–65.
40. Morales-Castañeda B, Zaldivar D, Cuevas E, Fausto F, Rodríguez A. A better balance in metaheuristic algorithms: does it exist? Swarm Evol Comput. 2020;54: 100671.