COMPLEX SYSTEM MODELING AND SIMULATION
ISSN 2096-9929 01/06 pp 261−281
Volume 3, Number 4, December 2023
DOI: 10.23919/CSMS.2023.0019

Harmony Search Algorithm Based on Dual-Memory Dynamic Search and Its Application on Data Clustering

Jinglin Wang, Haibin Ouyang*, Zhiyu Zhou, and Steven Li

Abstract: Harmony Search (HS) algorithm is highly effective in solving a wide range of real-world engineering optimization problems. However, it still has the problems such as being prone to local optima, low optimization accuracy, and low search efficiency. To address the limitations of the HS algorithm, a novel approach called the Dual-Memory Dynamic Search Harmony Search (DMDS-HS) algorithm is introduced. The main innovations of this algorithm are as follows: Firstly, a dual-memory structure is introduced to rank and hierarchically organize the harmonies in the harmony memory, creating an effective and selectable trust region to reduce approach blind searching. Furthermore, the trust region is dynamically adjusted to improve the convergence of the algorithm while maintaining its global search capability. Secondly, to boost the algorithm's convergence speed, a phased dynamic convergence domain concept is introduced to strategically devise a global random search strategy. Lastly, the algorithm constructs an adaptive parameter adjustment strategy to adjust the usage probability of the algorithm's search strategies, which aim to rationalize the abilities of exploration and exploitation of the algorithm. The results tested on the Computational Experiment Competition on 2017 (CEC2017) test function set show that DMDS-HS outperforms the other nine HS algorithms and the other four state-of-the-art algorithms in terms of diversity, freedom from local optima, and solution accuracy. In addition, applying DMDS-HS to data clustering problems, the results show that it exhibits clustering performance that exceeds the other seven classical clustering algorithms, which verifies the effectiveness and reliability of DMDS-HS in solving complex data clustering problems.

Key words: harmony search; dual-memory; dynamic search; optimization; data clustering

1 Introduction

The emergence of meta-heuristic search algorithm provides a powerful way to solve these complex engineering optimization problems. These algorithms are based on heuristic principles such as biology, nature, and social behavior, and seek potential optimal solutions by simulating processes such as natural evolution and swarm intelligence. They have the ability to search for high-dimensional, non-linear, and multimodal problems, and are usually able to obtain satisfactory solutions in a relatively short time. Due to the flexibility and adaptability of meta-heuristic search algorithms, they have been widely used in the field of engineering optimization, including but not limited to structural design[1−4], scheduling[5, 6], and path planning[7−10]. As a result, they have attracted a lot of attention in recent years, and simple meta-heuristic search algorithms have become increasingly important. Harmony Search (HS) algorithm is a music-inspired swarm search algorithm that draws inspiration from

• Jinglin Wang, Haibin Ouyang, and Zhiyu Zhou are with the School of Mechanical and Electric Engineering, Guangzhou University, Guangzhou 510006, China. E-mail: 2007700005@e.gzhu.edu.cn; oyhb1987@163.com.
• Steven Li is with the Graduate School of Business and Law, RMIT University, Melbourne 3000, Australia. E-mail: steven.li@rmit.edu.au.
∗ To whom correspondence should be addressed.
※ This article was recommended by Associate Editor Wenyin Gong.
Manuscript received: 2023-07-30; revised: 2023-10-06; accepted: 2023-10-16

music improvisation, which is proposed by Geem et al.[11] in 2001. In HS algorithm, the objective function is treated as musical composition composed of decision variables. By simulating the adjustment and coordination of musical notes, HS aims to obtain optimal solutions, or better "music". HS has several advantages, including simplicity of implementation, ease of parameter tuning, and relatively quick convergence compared to other optimization algorithms. It has found successful applications in various fields, such as feature selection[12], robot path planning[13], economic dispatch[14−16], shop scheduling[17−19], neural networks[20−24], and image processing[25−28]. Despite its successes, the HS algorithm still faces challenges, including slow convergence speed and weak local search capabilities, which can impact its optimization performance. To overcome these challenges, researchers have improved HS through parameter settings, search strategies, and ensembles with other optimization algorithms.

The HS algorithm has some limitations in terms of parameter configuration. Currently, contributions to parameter adaptive adjustment and fuzzy control have been made by scholars. For instance, Peraza et al.[29] used Interval 2 Fuzzy Logic System to dynamically adjust parameters in the HS algorithm, which changed the algorithm's global and local search capabilities. Shaqfa and Orbán[30] recorded occurrence probabilities of Harmony Memory Consideration Rate (HMCR) and Pitch Adjustment Rate (PAR) in generated and replaced solution vectors, redefining HMCR and PAR based on the current performance stage. They also introduced new parameters, HMCRmax and PARmin, to limit the working range of spontaneous probability for design variables[30]. Jeong et al.[31] proposed an advanced parameter-less version of HS to solve the parameter setting problem by using an improved Parameter-Setting-Free (PSF) scheme, reducing memory consumption, and improving efficiency. Valdez et al.[32] applied fuzzy logic during the algorithm execution and dynamically adjusted the main parameter HMCR using triangular membership functions. Ocak et al.[33] designed a version of HS with adaptive parameter variation for optimizing the Tuned Liquid Damper (TLD). They assigned initial values to the parameters (HMCR and BandWidth (BW)) of HS and gradually reduced these values with increasing iteration counts[33]. However, there are still challenges in current research, such as the requirement for extensive expertise and experience for parameter adjustment and the low adaptability of adjustment schemes that need to be addressed.

To raise the search effectiveness and optimization performance of the HS, scholars have proposed innovative methods. Boryczka and Szwarc[34] introduced a Modification rate (MOD) to improve harmony preservation and setting, enhancing the algorithm's effectiveness in solving the traveling salesman problem. Yi et al.[35] incorporated chaos into the HS algorithm by conducting parallel chaotic local searches from multiple starting points, reducing sensitivity to initial conditions and improving robustness. Doush et al.[17] improved the harmony memory using Nearest Neighbor (NN) and Modified Nawaz-Enscore-Ham (MNEH) techniques, exploring search space regions with different heuristic methods and enhancing global search capability. Wang et al.[36] transformed optimization variables into matrix form, utilizing a Conductor State Memory (CSM) to record time sequence constraints for scheduling problems. Li et al.[26] proposed an innovative harmony generation strategy that relies on explicit learning experiences, improving search efficiency and applying it to image segmentation with constraints on the search space. However, some enhancement strategies require significant computational resources and time, limiting their application to large-scale problems.

Scholars have explored combining strategies from different algorithms to achieve complementary optimization. Amini and Ghaderi[37] introduced dynamic weighting factors from the ant colony optimization algorithm and constant weighting factors based on structural system modal analysis into the HS to improve its convergence speed. Gheisarnejad[38] combined strategies from the cuckoo optimization algorithm with the HS algorithm by introducing intelligent laying and hybrid migration mechanisms to equilibrium the abilities of exploration and exploitation. Kayabekir et al.[39] integrated the capability of local search of flower pollination algorithm with the global search capability of the HS algorithm for optimizing structural systems with certain degrees of freedom. Radman[40] fused the HS with the Bi-directional Evolutionary Structural Optimization (BESO) algorithm, balancing the fast convergence of BESO and the strong global search capability of the HS to optimize the topology structure of cellular materials at the microscale. Gong et al.[41] combined the Tabu search algorithm with the HS, leveraging the Tabu search algorithm's performance in neighborhood search and the HS's advantages in global search to solve the row layout problem. While these studies have achieved certain results by combining strategies from different algorithms, determining trade-off ratios and interaction methods between different algorithms remains a challenging problem.

Many enhancements to the HS algorithm have not fully leveraged the information and experience stored in the harmony memory. Instead, typically only the best or worst harmony in the harmony memory is used, and a random adjustment is made to generate a new harmony. This approach has limitations because relying solely on the best or worst harmony can easily trap the population in local optima and make it difficult to escape from them. Therefore, exploring how to better utilize the individual harmonies in the Harmony Memory (HM) to get guide for the search process is worth investigating. In there, a new algorithm, harmony search algorithm based on dual-memory dynamic search, Dual-Memory Dynamic Search Harmony Search (DMDS-HS) algorithm, is proposed to improve the capability of optimization. This algorithm constructs two sets of harmonies with different ranks as candidate pools to determine the position and scope of the adaptive trust region for search. A nonlinear dynamic convergence domain adjustment is introduced to adjust the global search range in stages. Furthermore, appropriate adaptive variations are designed for parameters HMCR, PAR, and BW based on an improved search strategy. For the two sets of harmonies, one is the dominant memory, and the other is the archive memory. By arbitrarily selecting one dominant harmony and one archive harmony, a trusted search area is constructed for conducting the search, providing a certain directionality for individual evolution. The proposed algorithm is evaluated against various HS algorithms and heuristic algorithms for the Computational Experiment Competition on 2017 (CEC2017) benchmark function set. Results indicate that the proposed algorithm outperforms others in terms of solution accuracy, stability, and search capability, especially for complex high-dimensional problems. Furthermore, the DMDS-HS algorithm is applied to clustering problems and compared with other clustering algorithms, demonstrating its notable advantages in solving such problems.

The article's structure is outlined as follows. In Section 2, this paper provides a brief introduction to the HS algorithm, while Section 3 presents a detailed description of the proposed DMDS-HS algorithm. In Section 4, we will present numerical experiments and application results of the DMDS-HS algorithm, along with analysis. Furthermore, we will assess the computational intricacy of the enhanced algorithm. Lastly, a comprehensive summary of the entire paper will be provided in Section 5.

2 Harmony Search Algorithm

In HS, each solution is an *N*-dimensional vector and is collectively referred to as a "harmony". These harmonies are randomly generated at the initial stage and stored in the HM. The algorithm involves initialization, improvisation of new harmonies, and updating the harmony memory to optimize the process. The steps are as follows.

2.1 Initializing harmony memory

At the beginning of the algorithm, it is crucial to define the extent of the search space and generate an initial harmony memory within the extent. The search space is the set of feasible ranges for which there exists a globally optimal solution for each dimension of the decision variable. The initial harmony memory consists of a set of randomly generated harmony values. In each harmony, each decision variable represents a note, and the range of values of the note is determined by the search space.

2.2 Generating new harmonies

During the harmony update stage, it is necessary to utilize the information and experience saved in the HM to adjust each harmony. The harmony memory consists of a set of optimal and suboptimal solutions maintained throughout the search process, which guides the harmonies towards better solutions. The size of the HM is typically set to Harmony Memory Size (HMS), representing the number of harmonies. To generate new harmonies, a specific formula can be used.

$$x_{\text{new}} = \begin{cases} x_{\text{new}}(j) = x_a(j), r_1 < \text{HMCR}, r_2 > \text{PAR}; \\ x_{\text{new}}(j) = x_a(j) + r \times \text{BW}, r_1 < \text{HMCR}, r_2 < \text{PAR}; \\ x_{\text{new}}(j) = \text{LB}_j + r \times \left(\text{UB}_j - \text{LB}_j\right), r_1 > \text{HMCR} \end{cases} \tag{1}$$

where x_{new} represents the generated new harmony. $x_{\text{new}}(j)$ represents the *j*-th variable of x_{new}. x_a represents a randomly selected harmony in HM, and $x_a(j)$ represents the *j*-th variable of x_a. LB_j represents the lower boundary of the *j*-th variable. UB_j represents the upper boundary of the j-th variable. HMCR represents the harmony memory consideration rate. PAR represents the pitch adjustment rate, and BW represents the bandwidth size. r_1, r_2, and r represent random numbers between 0 and 1.

2.3 Updating the harmony memory

During the updating of HS, it is necessary to update the harmony memory based on the new harmony. If the adaptation value of the worst harmony in the HM is less than the newly generated harmony, it is replaced, and thus the HM is updated. Otherwise, no modification is made. In summary, the selection is made in the harmony by employing a greedy strategy, and the operations of generate new harmonies and update HM are repeated until the maximum number of iterations is reached.

3 Harmony Search Algorithm Based on Dual-Memory Dynamic Search

The current best solution in the HS algorithm represents only the best level among all current solutions and does not reflect the evolutionary direction of the overall level. Especially in high-dimensional complex optimization problems, if the current best solution is only a local optimum, it will be difficult to guide the creation of new harmonies and achieve the desired optimization results. Therefore, the HS tends to fall into a local optimum. To avoid this situation, multiple better solutions can be used to guide the generation of new solutions. Based on this idea, the DMDS-HS proposes four innovations: Firstly, it constructs a dual-memory system consisting of a dominant memory and an archive memory. Secondly, it constructs a dynamic trust region for search by selecting any combination of harmonies from the dominant and archive memories. Thirdly, it designs a stage-wise changing nonlinear dynamic convergence region. Finally, it introduces adaptive changing parameters HMCR, PAR, and BW. These innovations enable the algorithm to have better global optimization capability and faster convergence speed. At the end of this section, we introduce the calculation steps of DMDS-HS in detail, and the pseudocode of DMDS-HS is shown in Algorithm 1.

Algorithm 1 DMDS-HS

Initialization parameters.
Initialize UHM and LHM.
while $T < T_{\max}$ **do**
  Update the SHM： SHM $= \left\{x_1^{\text{best}}, x_2, x_{\text{HMS}-1}, x_{\text{HMS}}, x_{\text{SHM}}^{\text{mean}}\right\}$
  Update parameters HMCR, PAR, and BW by Eqs. (13)−(15).
  Update x^{ub} and x^{lb} by Eqs. (10) and (11).
  for each $j \in [1, D]$ **do**
    if $r_1 <$ HMCR **then**
      $t = \left(1 - \frac{T}{T_{\max}}\right)^{\frac{T}{T_{\max}}}$
      $\omega = 2 \times \text{sign}(r - 0.5) \times [\text{e}^{-\lambda t} - 1]$
      $x_j^{\text{new}} = x_j^{\text{SHM}} + \left(x_j^{\text{LHM}} - x_j^{\text{SHM}}\right) \times \omega$
      if $r_2 <$ PAR **then**
        $x_j^{\text{new}} = x_j^{\text{new}} \pm \text{rand} \times \text{BW}$
        rand is a random number in [0, 1]
      end if
    else
      $\begin{cases} x_j^{\text{new}} = \text{lb}_j + \text{rand} \times \left(\text{ub}_j - \text{lb}_j\right), \text{ if iter} \leqslant T_{\max}/2; \\ x_j^{\text{new}} = x_j^{\text{lb}} + \text{rand} \times \left(x_j^{\text{ub}} - x_j^{\text{lb}}\right), \text{ if iter} > T_{\max}/2; \end{cases}$
    end if
  end for
  Update the UHM and LHM.
  $T = T + 1$
end while

Note: UHM: upper-level harmony memory; LHM: lower-level harmony; SHM: senior harmony memory.

3.1 Dual harmony memory

The construction of a dual-memory harmony aims to provide various combinations of choices for generating new harmonies, increasing the diversity of generated harmonies, and reducing the likelihood of falling into a local optimal state. The initialization of the dual-memory harmony involves two harmonious memories of size HMS. The harmonies are sorted based on the fitness values from the best one to the worst one. The top HMS harmonies are allocated to the UHM, while the remaining HMS harmonies are allocated to the LHM. In each iteration, the best, second-best, worst, second-worst, and the mean of these four harmonies from the UHM are selected to form the SHM. The specific steps are as follows:

Initialize the sort:

$$\left\{x_1^{\text{best}}, x_2, x_3, \ldots, x_{\text{HMS}}, x_{\text{HMS}+1}, \ldots, x_{2\text{HMS}-1}, x_{2\text{HMS}}^{\text{worst}}\right\} \quad (2)$$

$$\text{UHM}: \left\{x_1^{\text{best}}, x_2, x_3, \ldots, x_{\text{HMS}}\right\} \quad (3)$$

$$\text{LHM}: \left\{x_{\text{HMS}+1}, \ldots, x_{2\text{HMS}-1}, x_{2\text{HMS}}^{\text{worst}}\right\} \quad (4)$$

$$\mathrm{SHM}:\left\{x_1^{\mathrm{best}}, x_2, x_{\mathrm{HMS}-1}, x_{\mathrm{HMS}}, x_{\mathrm{SHM}}^{\mathrm{mean}}\right\} \tag{5}$$

The updating method for the upper-level and lower-level memories involves greedy selection and archiving strategy. For the upper-level memory, a greedy selection approach is employed. If the new harmony's fitness value (x_{new}) exceeds that of the worst one in the upper-level memory (x_{HMS}), x_{new} replaces x_{HMS} in the upper-level memory. Simultaneously, x_{HMS} is moved to the lower-level memory, and the worst one in the lower-level memory ($x_{2\mathrm{HMS}}^{\mathrm{worst}}$) is removed to maintain the capacity of the lower-level memory. Consequently, the lower-level memory stores the harmonies that have been eliminated from the upper-level memory, thus also referred to the archiving memory. The specific updating process is illustrated in Fig. 1.

3.2 Dynamic trust region search

The purpose of constructing the dual-layer memory is to classify harmonies into different tiers, aiming to guide the evolution of harmonies in the lower-level memory with the influence of superior harmonies in the upper-level memory. The specific evolutionary strategy involves selecting one harmony from both the dominant memory and the archiving memory, using the dominant harmony as the center and the archiving harmony as the boundaries, to construct a trusted search region for exploration. The specific procedure is as follows:

$$x_j^{\mathrm{new}} = x_j^{\mathrm{SHM}} + \left(x_j^{\mathrm{LHM}} - x_j^{\mathrm{SHM}}\right) \times \omega \tag{6}$$

$$\begin{cases} \omega = 2 \times \mathrm{sign}(r - 0.5) \times \left[\mathrm{e}^{-\lambda t} - 1\right], \\ t = \left(1 - \dfrac{T}{T_{\max}}\right)^{\frac{T}{T_{\max}}} \end{cases} \tag{7}$$

where x_j^{new} represents the j-th variable of x_{new}, x_j^{SHM} represents the j-th variable of any harmony in SHM, x_j^{LHM} represents the j-th variable of any harmony in LHM, r and λ are random numbers in the range of [0, 1], T represents the total number of iterations completed so far, and $T_{\max}$ denotes the maximum allowed number of iterations for the computation. t is a number that decreases from 1 to 0 as the number of iterations increases. As t decreases, the value range of ω decreases from 1 to 0. ω is an important parameter that controls the generation range of x_j^{new}. The greater the value range of ω, the larger the value range of x_j^{new}, and vice versa. Therefore, from Fig. 2, it can be observed that as the iterations progress, the range of the trust region will converge towards x_j^{SHM}, and the value of x_j^{new} under the same λ will be closer to x_j^{SHM}. In the early stages of computation, the trust region has a large range, which helps reduce the risk of the population getting stuck in local optima. As the computation progresses, conducting a small-range search near the guided harmony contributes to improving the convergence speed.

3.3 Phase-wise nonlinear dynamic convergence region

The harmony search algorithm relies on three rules to govern the generation of new harmonies: harmony pitch selection, pitch adjustment, and random generation within the search range. Among these rules, the third rule assists the population in escaping local optima by introducing global random generation, particularly in the early stages of computation. However, in the later stages, the global optimum is typically located near the population, making it difficult for continued global random search to look for the global optimum. To enhance the likelihood of discovering the global optimum, targeted random search can be conducted in the later stages of computation by considering the current position of the population. This is achieved through the use of a stage-

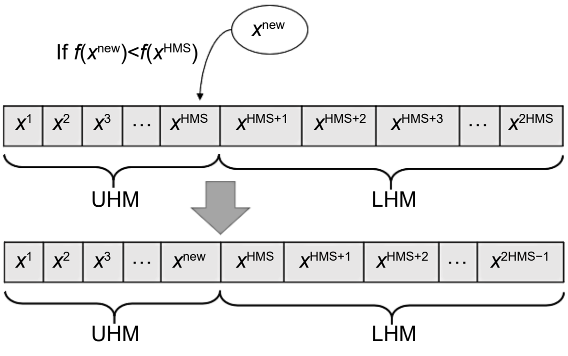


Fig. 1 UHM and LHM update process diagram.

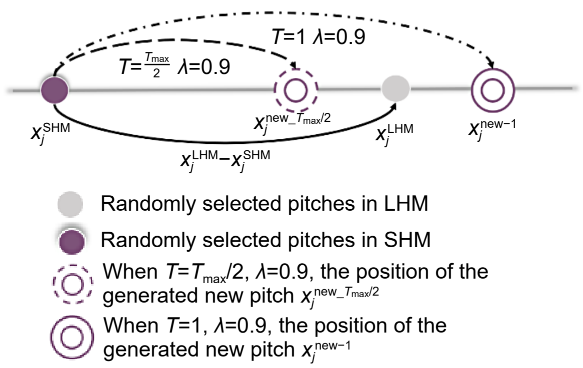


Fig. 2 Schematic diagram of dynamic search in trust region.

wise nonlinear dynamic convergence domain. The specific steps are as follows:

$$B_j^{\max} = \max(\text{SHM}) \tag{8}$$

$$B_j^{\min} = \min(\text{SHM}) \tag{9}$$

$$x_j^{\text{ub}} = x_j^{\text{ub}} + \left(B_j^{\max} - x_j^{\text{ub}}\right) \times (T/T_{\max})^2 \tag{10}$$

$$x_j^{\text{lb}} = x_j^{\text{lb}} + \left(B_j^{\min} - x_j^{\text{lb}}\right) \times (T/T_{\max})^2 \tag{11}$$

$$\begin{cases} x_j^{\text{new}} = \text{LB}_j + \text{rand} \times \left(\text{UB}_j - \text{LB}_j\right), \text{if iter} \leqslant T_{\max}/2; \\ x_j^{\text{new}} = x_j^{\text{lb}} + \text{rand} \times \left(x_j^{\text{ub}} - x_j^{\text{lb}}\right), \text{if iter} > T_{\max}/2 \end{cases} \tag{12}$$

where x_j^{ub} is the upper boundary of the dynamic convergence region. x_j^{lb} is the lower boundary of the dynamic convergence region. $B_j^{\max}$ stands for the maximum value of the j-th variable in SHM, and $B_j^{\min}$ expresses the minimum value of the j-th variable in SHM. And they denote the range of convergence domain in the j-th and dimensions, respectively. "rand" refers to a randomly generated number that falls within the interval of [0, 1]. Equation (12) shows how to generate new harmonies in a phase-wise nonlinear dynamic convergence region. When $\text{iter} \leqslant T_{\max}/2$, that is, the number of iterations is before half of the maximum number of iterations, new harmonies are still randomly generated in the global search domain like HS algorithm to increase the diversity of harmonies. When $\text{iter} > T_{\max}/2$, i.e., after half the calculation, the new harmony is generated in the range $[x_j^{\text{lb}}, x_j^{\text{ub}}]$. Equations (8)−(11) show that the range is centered on SHM and decreases with the number of iterations in a quadratic nonlinear manner. By surrounding the SHM region, the search range of the random search domain is narrowed, the probability of generating inferior harmony is reduced, and the solving efficiency and accuracy of the algorithm are improved.

3.4 Dynamic parameter adaptation

The parameters HMCR and PAR in the HS determine the probabilities of using the three rules throughout the computation process. Therefore, the settings of HMCR and PAR make a big difference on the harmony search algorithm. In order to set HMCR and PAR appropriately and adjust them during different computation stages, further research is needed. In the aforementioned DMDS-HS algorithm, improvements have been made to the first and third rules. To ensure the maximum effectiveness of these improved rules, new HMCR and PAR values should be designed to adjust the usage of these three rules effectively. To achieve a trade-off between exploration and exploitation abilities in the DMDS-HS algorithm, this study proposes a new nonlinear adaptive HMCR, which is used to adjust the usage of the first and third rules in a reasonable manner. Meanwhile, a linearly changing PAR and a logarithmic changing BW are employed. The specific formulas are as follows:

$$\text{HMCR} = \begin{cases} 0.5 + 1.0 \times \sqrt{T/T_{\max} \times (1 - T/T_{\max})}, \\ \quad \text{if } T \leqslant \dfrac{T_{\max}}{2}; \\ 0.8 + 0.4 \times \sqrt{T/T_{\max} \times (1 - T/T_{\max})}, \\ \quad \text{if } T > T_{\max}/2 \end{cases} \tag{13}$$

$$\text{PAR} = \text{PAR}_{\min} + (\text{PAR}_{\max} - \text{PAR}_{\min}) \times (T/T_{\max})^2 \tag{14}$$

$$\text{BW} = \text{BW}_{\max} \times \exp\left(\ln\left(\frac{\text{BW}_{\min}}{\text{BW}_{\max}}\right) \times \frac{T}{T_{\max}}\right) \tag{15}$$

3.5 Steps of the DMDS-HS algorithm

The complete steps of DMDS-HS are as follows:

Step 1: Initialization of NIGHS parameters. In this step, DMDS-HS algorithm parameters are defined, such as the number of decision variables (D), search upper and lower bounds for each variable (ub and lb), harmonic memory size (HMS), and maximum number of iterations ($T_{\max}$).

Step 2: Initialization of the UHM and LHM. The HM generated by the initial is sorted according to the fitness value and divided into UHM and LHM. See Formulas (2)–(4) for detailed methods.

Step 3: Update the SHM, HMCR, PAR, BW, x^{ub}, and x^{lb}. In this step we update the SHM, HMCR, PAR, BW, x^{ub}, and x^{lb} through Formula (5) and Eqs. (8)–(11) and (13)–(15).

Step 4: Improvisation of a new harmony. In this step, a new harmony (x^{new}) is created through a dynamic trust domain search and a phase-wise nonlinear dynamic convergence region (Lines 7−20 of Algorithm 1).

Step 5: Update the UHM and LHM. If the fitness value of the new harmony (x^{new}) is better than the fitness value of the worst harmony in UHM (x^{worst}), then the worst harmony in UHM will be replaced by the new harmony, and x^{worst} will replace the worst harmony in LHM.

Step 6: Check the termination criterion. If the number of the current iteration (T) is less than the maximum number of iterations ($T_{\max}$), then Steps 3 and

4 are repeated. Otherwise, the optimization process stops.

4 Comparison and Analysis of Experimental Results

The conducted experiments utilized a system that comprised an Intel(R) Xeon(R) processor with a clock speed of 2.76 GHz, 36 processors, 160 GB of RAM, and ran on the Windows 10 operating system. The programming implementation was done using MATLAB R2015b. To validate the capability of the DMDS-HS, we selected 9 HS algorithm variants for comparison: HS[11], SGHS[42], IHS[43], GHS[44], NGHS[45], IGHS[46], LHS[47], IMGHS[48], and ID-HS-LDD[49]. We also tested these algorithms on the well-known CEC2017 benchmark function set. In all experiments, each algorithm ran 51 times on the 30 functions in the CEC2017 benchmark function set. The experiments were conducted for three different dimensions: D =10, 30, and 50. The search space for all test functions encompassed the interval of [−100, 100]. The upper limit for the of evaluations of functions ($T_{\max}$) was determined as 10 000 × D based on the benchmark rules, and an error value below 10^{-8} was treated as 0. In addition to comparing the improved versions of HS, we also compared DMDS with four other cutting-edge heuristic algorithms, namely SLWCHOA[50], IWOA[51], HGWO[52], and GWO[53], under 30D conditions. To facilitate statistical analysis of the experimental results, we use the terms of error values to presented the results. The parameter settings for each algorithm are shown in Table 1.

Table 1 Parameters setting.

Algorithm	Parameter
HS	HMS=5, HMCR=0.9, PAR=0.3, BW=0.01.
IHS	HMS=5,$BW_{\min} = 0.0001$,$BW_{\max} = (UB-LB)/20$, HMCR=0.9, $PAR_{\min} = 0.01$, $PAR_{\max} = 0.99$.
GHS	HMS=5, HMCR=0.9, $PAR_{\min} = 0.01$,$PAR_{\max} = 0.9$.
SGHS	HMS=5, HMCR=0.98, PAR=0.9, $BW_{\min} = 0.0005$, $BW_{\max} = (UB-LB)/10$, lp=100.
NGHS	HMS=5, $P_m = 0.005$.
IGHS	HMS=5, $P_m = 0.005$, PAR=0.4.
LHS	HMS=5, HMCR=0.99.
IMGHS	HMS=5, HMCR=0.9, PAR=0.3, BW=0.01, $P_m = 0.005$, $\mu_1 = 0.7$, $\mu_2 = 0.3$.
ID-HS-LDD	HMS=30, $HMCR_{\min} = 0.3$, $HMCR_{\max} = 0.99$, $PAR_{\min} = 0.3$, $PAR_{\max} = 0.99$.
DMDS-HS	HMS=5, $PAR_{\min} = 0.01$, $PAR_{\max} = 0.99$, $BW_{\min} = 0.0001$, $BW_{\max} = (UB-LB)/20$.

4.1 Comparison of DMDS-HS under the same evaluation budget

By conducting 51 independent experiments for each function of each algorithm, we obtained the mean and standard deviation (std) of the experimental results under conditions D =10, 30, and 50. The best result for each group is highlighted in bold. To compare the performance of different algorithms, we used non-parametric tests (Mann-Whitney U test) to verify whether there are statistically significant differences in performance between DMDS-HS and other algorithms. The Mann-Whitney U test, also known as the Mann-Whitney-Wilcoxon test[54, 55], is a non-parametric rank-based test method used to compare differences between different groups. It has been widely used for performance comparison of heuristic algorithms[17]. In the non-parametric test, the symbol "+" indicates that the overall result of the DMDS-HS is better than the specific algorithm on a particular function; the symbol "–" indicates that the overall result of the DMDS-HS algorithm is worse than the specific algorithm on a particular function; the symbol "=" indicates that the overall result of the DMDS-HS algorithm is similar to the specific algorithm on a particular function. The final statistical results are presented in Table 2 in the format of "+/–/=".

According to the results in Tables 3−8, we performed nonparametric tests on 51 instances to independently validate the results when D = 10, 30, and 50. In the case of D = 10, the DMDS-HS algorithm is significantly superior to HS, IHS, GHS, SGHS, NGHS, IGHS, LHS, IMGHS, and IDHS-LDD algorithms in 29, 24, 28, 28, 29, 29, 27, 30, and 13 of the 30 functions, respectively. Similarly, when D = 30, the DMDS-HS algorithm outperforms the HS, IHS, GHS, SGHS, NGHS, IGHS, LHS, IMGHS, and ID-HS-LDD algorithms in 28, 27, 29, 30, 29, 30, 23, 26, and 20 of the 30 functions, respectively. And in the high-dimensional case (D = 50), the DMDS-HS algorithm also performs well. It outperforms the HS, IHS, GHS, SGHS, NGHS, IGHS, LHS, IMGHS, and ID-HS-LDD algorithms in 26, 26, 28, 30, 28, 30, 25, 26, and 24 of the 30 functions, respectively. As can be seen from the comparative notation in Tables 3−9, although the DMDS-HS algorithm does not give satisfactory results on functions F10, F12, and F18, it achieves the best results on the vast majority of the other functions. In low-dimensional problems (D = 10), the DMDS-HS algorithm does not perform as well as the ID-HS-LDD algorithm. However, as the dimensionality increases,

Table 2 Statistical results of DMDS-HS and other algorithm groups on data clustering problems.

Dataset	Statistical index	K-means	K-means++	GA	PSO	DE	SCA	HS	DMDS-HS
IRIS	Mean	1.0234×10^{2}	9.8344×10^{1}	1.3228×10^{2}	1.1835×10^{2}	1.0752×10^{2}	1.2851×10^{2}	9.6860×10^{1}	**9.6676×10^{1}**
	Std	1.0235×10^{1}	5.0463	5.9674	1.2251×10^{1}	1.3044×10^{1}	5.3505	4.9423×10^{-1}	**1.4520×10^{-1}**
	Sign	+	+	+	+	+	+	+	—
CMC	Mean	5.5438×10^{3}	5.5436×10^{3}	6.3188×10^{3}	6.3332×10^{3}	5.8357×10^{3}	6.5907×10^{3}	6.9137×10^{3}	**5.5322×10^{3}**
	Std	1.5789	1.5511	1.6593×10^{2}	4.0394×10^{2}	2.4215×10^{2}	2.9741×10^{2}	6.4980×10^{2}	**6.8641×10^{-7}**
	Sign	+	+	+	+	+	+	+	—
Glass	Mean	2.2567×10^{2}	**2.2463×10^{2}**	4.2007×10^{2}	3.8549×10^{2}	3.2868×10^{2}	3.6654×10^{2}	3.2927×10^{2}	2.3729×10^{2}
	Std	1.2569×10^{1}	1.1859×10^{1}	1.9849×10^{1}	3.3833×10^{1}	3.5507×10^{1}	1.3814×10^{1}	6.5071×10^{1}	**6.0636**
	Sign	−	−	+	+	+	+	+	—
Balance	Mean	1.4267×10^{3}	1.4276×10^{3}	1.4402×10^{3}	1.4277×10^{3}	**1.4244×10^{3}**	1.4389×10^{3}	1.4273×10^{3}	1.4252×10^{3}
	Std	3.1110	3.8407	4.0364	2.1860	**2.0515**	2.9503	2.8966	1.3805
	Sign	+	+	+	+	−	+	+	—
Wine	Mean	1.7166×10^{4}	1.7423×10^{4}	1.6638×10^{4}	1.7930×10^{4}	1.6484×10^{4}	1.6556×10^{4}	1.9303×10^{4}	**1.6293×10^{4}**
	Std	8.7435×10^{2}	9.3234×10^{2}	1.2506×10^{2}	8.1625×10^{2}	1.9947×10^{2}	8.0615×10^{1}	1.3714×10^{3}	**7.9050×10^{-1}**
	Sign	+	+	+	+	+	+	+	—
Aggregation	Mean	2.7726×10^{3}	2.7697×10^{3}	3.1337×10^{3}	3.0234×10^{3}	2.7708×10^{3}	3.1040×10^{3}	3.1760×10^{3}	**2.7411×10^{3}**
	Std	6.5213×10^{1}	**5.2785×10^{1}**	7.2205×10^{1}	1.8300×10^{2}	6.9682×10^{1}	5.8419×10^{1}	2.0581×10^{2}	5.5920×10^{1}
	Sign	+	+	+	+	+	+	+	—
Vowel	Mean	**1.3925×10^{3}**	1.3934×10^{3}	1.8181×10^{3}	1.9411×10^{3}	1.8489×10^{3}	1.8831×10^{3}	1.8345×10^{3}	1.4666×10^{3}
	Std	**8.6113**	1.0904×10^{1}	2.7410×10^{1}	6.4071×10^{1}	1.2761×10^{2}	2.9419×10^{1}	9.5115×10^{1}	1.8239×10^{1}
	Sign	−	−	+	+	+	+	+	—
Compound	Mean	1.1470×10^{3}	1.1513×10^{3}	1.2594×10^{3}	1.2102×10^{3}	1.1008×10^{3}	1.2370×10^{3}	1.2416×10^{3}	**1.0719×10^{3}**
	Std	7.2641×10^{1}	7.8623×10^{1}	3.2775×10^{1}	7.1360×10^{1}	5.4181×10^{1}	2.6151×10^{1}	7.3336×10^{1}	**1.8900×10^{1}**
	Sign	+	+	+	+	−	+	+	—
Cancer	Mean	2.9877×10^{3}	2.9881×10^{3}	4.3903×10^{3}	4.5149×10^{3}	3.2164×10^{3}	3.2356×10^{3}	5.0479×10^{3}	**2.9644×10^{3}**
	Std	7.3925×10^{-1}	5.8802×10^{-1}	2.3740×10^{2}	6.4333×10^{2}	1.9787×10^{2}	4.2978×10^{1}	5.8233×10^{2}	**3.2000×10^{-3}**
	Sign	+	+	+	+	+	+	+	—
Sonar	Mean	**2.3505×10^{2}**	2.3506×10^{2}	3.2647×10^{2}	3.3568×10^{2}	2.8719×10^{2}	3.3855×10^{2}	2.3757×10^{2}	2.4739×10^{2}
	Std	**1.7159×10^{-1}**	1.8616×10^{-1}	6.7534	1.9232×10^{1}	1.2761×10^{1}	5.8943	5.1230	5.3927
	Sign	−	−	+	+	+	+	−	—
+/−/=		7/3/0	7/3/0	10/0/0	10/0/0	7/2/0	10/0/0	9/1/0	—

Note: CMC is the contraceptive method choice dataset.

the DMDS-HS algorithm significantly outperforms the ID-HS-LDD algorithm. Further observing the data in Table 9, the DMDS-HS algorithm statistically outperforms the SLWCHOA, IWOA, HGWO, and GWO algorithms in 30, 29, 30, and 29 of the 30 functions when D = 30 and all algorithms have the same number of function evaluations. In summary, the DMDS-HS algorithm outperforms the other nine HS algorithms as well as the four advanced heuristics. These results clearly demonstrate the superiority of the DMDS-HS algorithm in different dimensions.

According to the analysis of the experimental results, we can gain clearer insights into the performance of the DMDS-HS algorithm. The iteration plot in Fig. 3 reveals that the DMDS-HS algorithm utilizes adaptive variation design of HMCR. Initially, it has a small HMCR value, which increases the probability of Rule 3 (see the third line in Eq. (1)) for global random search. Consequently, the algorithm exhibits slower convergence at the beginning. However, as the iteration progresses, HMCR gradually increases, and Rule 3 performs local search, leading to accelerated convergence and higher-precision solutions compared to the other nine algorithms. Notably, in the case of the function F20, the DMDS-HS algorithm demonstrates a particularly strong ability to escape local optima. This can be attributed to the construction of the dual-memory structure and dynamic trust region, which provide a richer diversity for generating new harmonies in the DMDS-HS algorithm. The results presented in

Table 3 Experimental results of HS, IHS, GHS, SGHS, and NGHS in CEC2017, when D = 10.

Function	HS			IHS			GHS			SGHS			NGHS		
	Mean	Std	Sign	Mean	Std	Sign	Mean	Std	Sign	Mean	Std	Sign	Mean	Std	Sign
F1	3.08267×10^{3}	3.15712×10^{3}	+	2.02048×10^{3}	2.23830×10^{3}	−	2.47874×10^{6}	2.02361×10^{6}	+	1.79548×10^{9}	8.22628×10^{8}	+	8.71528×10^{5}	9.61851×10^{5}	+
F2	5.80438	3.68125×10^{1}	+	1.75997×10^{-4}	1.54147×10^{-4}	+	5.74012×10^{4}	1.70538×10^{5}	+	6.00596×10^{8}	6.39031×10^{8}	+	1.79928×10^{6}	7.04627×10^{6}	+
F3	1.88152×10^{2}	2.40265×10^{2}	+	**0.00000**	**0.00000**	=	4.78126×10^{2}	3.25676×10^{2}	+	8.77812×10^{3}	2.42916×10^{3}	+	2.86004×10^{4}	1.42385×10^{4}	+
F4	7.90070	1.51119×10^{1}	+	1.43752	8.28746×10^{-1}	+	2.02566×10^{1}	3.10984×10^{1}	+	1.65968×10^{2}	5.53968×10^{1}	+	3.05252×10^{1}	3.71431×10^{1}	+
F5	1.03022×10^{1}	3.86819	+	1.12221×10^{1}	4.55692	+	1.30928×10^{1}	4.99566	+	6.80026×10^{1}	8.07514	+	2.19133×10^{1}	9.67173	+
F6	9.40332×10^{-2}	5.63271×10^{-2}	+	8.17156×10^{-5}	2.33630×10^{-5}	+	1.00965	3.94227×10^{-1}	+	3.47450×10^{1}	5.86177	+	3.71640×10^{-1}	3.72645×10^{-1}	+
F7	2.61333×10^{1}	7.30730	+	2.82342×10^{1}	5.99328	+	3.06434×10^{1}	7.06069	+	1.58600×10^{2}	2.65216×10^{1}	+	3.74618×10^{1}	9.54521	+
F8	1.19286×10^{1}	4.33330	+	1.17314×10^{1}	4.28119	+	1.30409×10^{1}	4.30791	+	6.55719×10^{1}	6.53124	+	2.24001×10^{1}	7.63109	+
F9	7.30069	8.24583	+	3.16630	1.25032×10^{1}	=	9.09435	1.19608×10^{1}	+	8.21185×10^{2}	1.78633×10^{2}	+	3.92980×10^{1}	5.02997×10^{1}	+
F10	2.94470×10^{2}	**1.44130×10^{2}**	+	3.32262×10^{2}	1.58976×10^{2}	+	3.77330×10^{2}	1.74573×10^{2}	+	1.39644×10^{3}	1.75411×10^{2}	+	6.33411×10^{2}	2.38277×10^{2}	+
F11	8.77882	4.89609	+	7.19997	3.79084	+	5.35035×10^{1}	5.30578×10^{1}	+	2.73207×10^{2}	8.20713×10^{1}	+	1.18114×10^{3}	2.10469×10^{3}	+
F12	2.94494×10^{4}	3.84189×10^{4}	+	1.82189×10^{4}	1.69821×10^{4}	+	1.67441×10^{5}	1.57411×10^{5}	+	4.44834×10^{7}	2.70338×10^{7}	+	2.82731×10^{6}	3.17243×10^{6}	+
F13	1.12739×10^{4}	1.00112×10^{4}	+	9.38800×10^{3}	1.01283×10^{4}	+	1.05650×10^{4}	1.14507×10^{4}	+	1.09866×10^{5}	1.07497×10^{5}	+	1.22798×10^{4}	1.40209×10^{4}	+
F14	4.35098×10^{3}	7.00665×10^{3}	+	3.53054×10^{3}	5.52295×10^{3}	+	1.12528×10^{3}	2.14718×10^{3}	+	2.47003×10^{2}	1.22135×10^{2}	+	7.60339×10^{3}	7.55409×10^{3}	+
F15	7.05973×10^{3}	8.47177×10^{3}	+	3.96450×10^{3}	5.88346×10^{3}	+	1.78851×10^{3}	1.83607×10^{3}	+	2.24962×10^{3}	1.17225×10^{3}	+	7.50622×10^{3}	8.50676×10^{3}	+
F16	1.05535×10^{2}	9.65878×10^{1}	+	1.10388×10^{2}	1.05237×10^{2}	+	1.25015×10^{2}	9.20700×10^{1}	+	2.34731×10^{2}	7.64903×10^{1}	+	3.09728×10^{2}	1.62427×10^{2}	+
F17	1.42672×10^{1}	1.37213×10^{1}	+	1.08355×10^{1}	1.54794×10^{1}	−	1.63495×10^{1}	1.23646×10^{1}	+	1.19549×10^{2}	2.70080×10^{1}	+	7.42113×10^{1}	7.08386×10^{1}	+
F18	1.00535×10^{4}	8.60780×10^{3}	−	1.00876×10^{4}	8.79564×10^{3}	−	6.56033×10^{3}	**5.66197×10^{3}**	−	2.95495×10^{5}	3.16249×10^{5}	+	1.20210×10^{4}	1.15406×10^{4}	−
F19	6.97982×10^{3}	9.03186×10^{3}	+	4.29688×10^{3}	5.98765×10^{3}	+	3.76203×10^{3}	4.12997×10^{3}	+	3.43952×10^{3}	2.85204×10^{3}	+	1.04376×10^{4}	9.05478×10^{3}	+
F20	5.14106	5.93167	+	5.64000	6.09823	+	9.54624	**4.06978**	+	1.24103×10^{2}	2.53600×10^{1}	+	1.56987×10^{1}	1.12534×10^{1}	+
F21	2.15970×10^{2}	2.46386×10^{1}	+	2.15779×10^{2}	2.47473×10^{1}	+	1.60291×10^{2}	3.46208×10^{1}	−	1.40839×10^{2}	**1.38751×10^{1}**	−	2.30401×10^{2}	3.93556×10^{1}	+
F22	2.17538×10^{2}	2.72685×10^{2}	+	2.53963×10^{2}	2.99385×10^{2}	+	1.07533×10^{2}	4.11863×10^{1}	+	2.24881×10^{2}	6.37436×10^{1}	+	4.64269×10^{2}	5.24633×10^{2}	+
F23	3.20288×10^{2}	7.87188	+	3.19176×10^{2}	7.42331	+	3.20954×10^{2}	6.68562	+	3.68481×10^{2}	4.03948×10^{1}	+	3.37150×10^{2}	2.75692×10^{1}	+
F24	3.42824×10^{2}	6.26015×10^{1}	+	3.46153×10^{2}	5.62316×10^{1}	+	3.31803×10^{2}	5.29542×10^{1}	+	2.91762×10^{2}	4.20194×10^{1}	−	3.65204×10^{2}	9.08090×10^{1}	+
F25	4.37933×10^{2}	2.06878×10^{1}	+	4.34445×10^{2}	2.08531×10^{1}	−	4.34749×10^{2}	3.10982×10^{1}	+	5.39560×10^{2}	4.20775×10^{1}	+	4.28417×10^{2}	5.34264×10^{1}	+
F26	6.27976×10^{2}	5.01689×10^{2}	+	7.49840×10^{2}	5.03972×10^{2}	+	3.97888×10^{2}	1.53586×10^{2}	+	7.16943×10^{2}	9.29932×10^{1}	+	9.60091×10^{2}	6.09590×10^{2}	+
F27	4.00827×10^{2}	1.55220×10^{1}	+	4.02273×10^{2}	1.26757×10^{1}	+	3.99443×10^{2}	5.78766	+	4.41930×10^{2}	9.95095	+	4.36958×10^{2}	3.41907×10^{1}	+
F28	4.77650×10^{2}	1.17587×10^{2}	+	4.41310×10^{2}	1.41621×10^{2}	+	4.26637×10^{2}	**4.32428×10^{1}**	+	5.77070×10^{2}	5.97865×10^{1}	+	5.16094×10^{2}	1.24708×10^{2}	+
F29	2.73683×10^{2}	2.29298×10^{1}	+	2.71769×10^{2}	2.36215×10^{1}	+	2.84997×10^{2}	3.52946×10^{1}	+	3.95382×10^{2}	3.10260×10^{1}	+	3.86239×10^{2}	6.20655×10^{1}	+
F30	3.60643×10^{5}	4.62597×10^{5}	+	6.36530×10^{4}	1.91906×10^{5}	+	4.36509×10^{5}	3.84545×10^{5}	+	3.13007×10^{6}	1.47789×10^{6}	+	1.11967×10^{6}	1.11655×10^{6}	+
+/−/=	**29/1/0**			**24/4/2**			**28/2/0**			**28/2/0**			**29/1/0**		

Table 4 Experimental results of IGHS, LHS, IMGHS, ID-HS-LDD, and DMDS-HS in CEC2017, when D = 10.

Function	IGHS			LHS			IMGHS			ID-HS-LDD			DMDS-HS	
	Mean	Std	Sign	Mean	Std	Sign	Mean	Std	Sign	Mean	Std	Sign	Mean	Std
F1	2.41270×10^{8}	2.07522×10^{8}	+	2.36956×10^{3}	2.83703×10^{3}	−	3.09471×10^{3}	3.02672×10^{3}	+	**1.49044×10^{3}**	**1.49420×10^{3}**	−	2.37144×10^{3}	2.36827×10^{3}
F2	9.58121×10^{6}	4.21978×10^{7}	+	3.56290×10^{-3}	3.49658×10^{-3}	+	5.26785×10^{-3}	5.06979×10^{-3}	+	1.60789×10^{-4}	2.78339×10^{-4}	+	**1.82547×10^{-5}**	**2.04600×10^{-5}**
F3	2.49292×10^{2}	2.11534×10^{2}	+	1.27240	3.67158	+	3.25809×10^{-4}	6.11916×10^{-4}	+	**0.00000**	**0.00000**	=	**0.00000**	**0.00000**
F4	4.17160×10^{1}	3.49526×10^{1}	+	1.33768×10^{1}	2.39465×10^{1}	+	8.60795	2.16745×10^{1}	+	**2.40181×10^{-2}**	**5.60287×10^{-3}**	−	6.27446×10^{-2}	2.24478×10^{-2}
F5	2.26633×10^{1}	9.12563	+	1.58329×10^{1}	7.13135	+	2.63956×10^{1}	9.67995	+	1.31861×10^{1}	5.05176	+	**6.31443**	**3.14723**
F6	5.50027	2.40212	+	7.80310×10^{-4}	3.19826×10^{-3}	−	5.57477×10^{-1}	6.05081×10^{-1}	+	1.51053×10^{-3}	2.79856×10^{-3}	+	**4.08675×10^{-5}**	**8.24116×10^{-6}**
F7	3.26858×10^{1}	1.18262×10^{1}	+	3.22193×10^{1}	7.77849	+	4.07124×10^{1}	1.06841×10^{1}	+	2.91377×10^{1}	1.23091×10^{1}	+	**1.50450×10^{1}**	**2.05173**
F8	1.53972×10^{1}	6.25430	+	1.94114×10^{1}	7.51800	+	2.89708×10^{1}	9.85285	+	1.38863×10^{1}	1.02020×10^{1}	+	**5.05553**	**1.88000**
F9	5.19856×10^{1}	2.95366×10^{1}	+	8.55880	1.36437×10^{1}	+	8.32732×10^{1}	8.19982×10^{1}	+	5.25130×10^{-1}	9.49713×10^{-1}	=	**0.00000**	**0.00000**
F10	4.71939×10^{2}	2.23624×10^{2}	+	5.55004×10^{2}	2.04494×10^{2}	+	7.47702×10^{2}	2.23164×10^{2}	+	9.85780×10^{2}	4.62680×10^{2}	+	**2.52684×10^{2}**	1.90184×10^{2}
F11	3.13986×10^{2}	6.06273×10^{2}	+	1.12386×10^{1}	6.10987	+	1.56065×10^{1}	9.25982	+	5.19403	3.77187	+	**1.99836**	**1.59068**
F12	1.08719×10^{6}	1.14823×10^{6}	+	5.53289×10^{5}	8.95794×10^{5}	+	1.41087×10^{4}	1.05421×10^{4}	+	**8.63134×10^{3}**	**6.61335×10^{3}**	−	1.41579×10^{4}	1.37466×10^{4}
F13	3.91759×10^{4}	1.25625×10^{5}	+	1.13857×10^{4}	9.73178×10^{3}	+	1.10416×10^{4}	1.03786×10^{4}	+	8.18726×10^{3}	**6.58040×10^{3}**	+	**6.77897×10^{3}**	7.02299×10^{3}
F14	3.68551×10^{3}	5.64628×10^{3}	+	1.48969×10^{3}	2.38856×10^{3}	+	7.24938×10^{3}	7.75936×10^{3}	+	**1.76603×10^{1}**	**1.88951×10^{1}**	−	6.14556×10^{2}	1.55001×10^{3}
F15	5.93995×10^{3}	5.94124×10^{3}	+	5.42424×10^{3}	6.68365×10^{3}	+	6.49447×10^{3}	7.76793×10^{3}	+	**2.06679×10^{2}**	**2.96593×10^{2}**	−	2.52331×10^{3}	5.61982×10^{3}
F16	1.76265×10^{2}	1.16741×10^{2}	+	2.01322×10^{2}	1.29138×10^{2}	+	2.80203×10^{2}	1.58272×10^{2}	+	3.22561×10^{1}	5.85561×10^{1}	−	**2.67320×10^{1}**	**4.50142×10^{1}**
F17	3.05572×10^{1}	2.53851×10^{1}	+	3.31454×10^{1}	3.71508×10^{1}	+	7.35175×10^{1}	7.76747×10^{1}	+	**7.05606**	1.05866×10^{1}	−	9.64341	**8.64729**
F18	6.12015×10^{4}	3.84997×10^{5}	−	7.04746×10^{3}	6.56665×10^{3}	−	1.53902×10^{4}	1.09883×10^{4}	+	**5.08955×10^{3}**	5.78692×10^{3}	−	1.19975×10^{4}	9.51046×10^{3}
F19	1.69988×10^{4}	5.47126×10^{4}	+	7.25405×10^{3}	7.32174×10^{3}	+	7.22472×10^{3}	8.56090×10^{3}	+	**9.47368×10^{2}**	**1.32192×10^{3}**	−	3.52451×10^{3}	6.65120×10^{3}
F20	1.93396×10^{1}	1.15777×10^{1}	+	7.26303	6.92265	+	1.52943×10^{1}	1.08851×10^{1}	+	3.91575	5.27572	+	**3.33188**	5.04917
F21	1.76438×10^{2}	5.74993×10^{1}	+	1.80893×10^{2}	6.24587×10^{1}	+	2.31306×10^{2}	4.17246×10^{1}	+	**1.04729×10^{2}**	2.16979×10^{1}	−	1.72304×10^{2}	5.10580×10^{1}
F22	1.25988×10^{2}	2.48496×10^{1}	+	1.03311×10^{2}	1.29794×10^{1}	+	5.05643×10^{2}	5.57769×10^{2}	+	**9.41438×10^{1}**	2.43830×10^{1}	+	1.00652×10^{2}	**4.57453×10^{-1}**
F23	3.26958×10^{2}	1.34775×10^{1}	+	3.27792×10^{2}	1.00152×10^{1}	+	3.47280×10^{2}	2.57773×10^{1}	+	3.14458×10^{2}	5.90840	+	**3.09093×10^{2}**	**3.51012**
F24	3.37738×10^{2}	6.16362×10^{1}	+	3.58580×10^{2}	6.93914×10^{1}	+	3.50566×10^{2}	1.01774×10^{2}	+	**1.09475×10^{2}**	4.73759×10^{1}	−	3.37448×10^{2}	**4.10459**
F25	4.56599×10^{2}	**1.51972×10^{1}**	+	4.34544×10^{2}	2.14359×10^{1}	+	4.30458×10^{2}	4.44756×10^{1}	+	**4.23968×10^{2}**	2.30028×10^{1}	−	4.31429×10^{2}	2.39254×10^{1}
F26	4.80735×10^{2}	2.17336×10^{2}	+	5.61107×10^{2}	4.04927×10^{2}	+	9.10846×10^{2}	5.84943×10^{2}	+	**2.86668×10^{2}**	8.80576×10^{1}	−	3.31217×10^{2}	**3.68988×10^{1}**
F27	4.05837×10^{2}	6.54802	+	4.16949×10^{2}	2.33358×10^{1}	+	4.31630×10^{2}	2.66433×10^{1}	+	3.96491×10^{2}	3.50117	+	**3.92539×10^{2}**	**2.61939**
F28	4.75831×10^{2}	1.18422×10^{2}	+	5.00986×10^{2}	1.40924×10^{2}	+	4.80813×10^{2}	1.60281×10^{2}	+	**3.13140×10^{2}**	8.89617×10^{1}	−	4.16487×10^{2}	1.47938×10^{2}
F29	2.93200×10^{2}	3.10689×10^{1}	+	3.32814×10^{2}	5.69340×10^{1}	+	3.60718×10^{2}	7.81436×10^{1}	+	2.72107×10^{2}	2.18049×10^{1}	+	**2.48062×10^{2}**	**1.11204×10^{1}**
F30	4.10201×10^{5}	4.20446×10^{5}	+	4.09162×10^{5}	5.37025×10^{5}	+	2.74120×10^{5}	4.63459×10^{5}	+	**4.09874×10^{3}**	**9.27012×10^{3}**	−	2.27259×10^{5}	3.86120×10^{5}
+/−/=	29/1/0			27/3/0			30/0/0			13/15/2			—	

Table 5 Experimental results of HS, IHS, GHS, SGHS, and NGHS in CEC2017, when *D* = 30.

Function	HS			IHS			GHS			SGHS			NGHS		
	Mean	Std	Sign	Mean	Std	Sign	Mean	Std	Sign	Mean	Std	Sign	Mean	Std	Sign
F1	6.51296×10^{3}	6.46223×10^{3}	+	$\mathbf{3.78845\times10^{3}}$	$\mathbf{4.41120\times10^{3}}$	+	2.20174×10^{8}	7.98512×10^{7}	+	4.18738×10^{10}	9.24275×10^{9}	+	2.15590×10^{5}	1.43238×10^{5}	+
F2	6.31301×10^{12}	1.42252×10^{13}	+	1.27255×10^{11}	3.07016×10^{11}	+	2.34888×10^{24}	1.63966×10^{25}	+	4.38395×10^{39}	1.22232×10^{40}	+	1.02379×10^{21}	7.31130×10^{21}	+
F3	5.95806×10^{3}	2.24778×10^{3}	+	5.55918×10^{3}	2.75806×10^{3}	+	3.15331×10^{4}	6.61569×10^{3}	+	1.08611×10^{5}	1.35630×10^{4}	+	1.26404×10^{5}	4.57539×10^{4}	+
F4	9.64482×10^{1}	3.59972×10^{1}	+	9.16420×10^{1}	2.88097×10^{1}	+	1.77818×10^{2}	3.51998×10^{1}	+	7.87598×10^{3}	2.27704×10^{3}	+	1.13463×10^{2}	4.16975×10^{1}	+
F5	6.78828×10^{1}	1.97519×10^{1}	+	7.14685×10^{1}	$\mathbf{1.59307\times10^{1}}$	+	1.21278×10^{2}	2.02766×10^{1}	+	4.16056×10^{2}	2.59020×10^{1}	+	1.24554×10^{2}	3.30784×10^{1}	+
F6	3.11350×10^{-1}	9.76051×10^{-2}	+	2.94054×10^{-2}	4.01052×10^{-2}	+	5.58262×10	1.12659×10	+	8.13474×10^{1}	6.66267×10	+	9.28427×10^{-2}	7.78629×10^{-2}	+
F7	1.20646×10^{2}	2.22739×10^{1}	+	1.34187×10^{2}	2.29275×10^{1}	+	2.20183×10^{2}	2.88561×10^{1}	+	1.31889×10^{3}	1.89046×10^{2}	+	1.54689×10^{2}	3.12202×10^{1}	+
F8	7.05118×10^{1}	1.86494×10^{1}	+	7.46720×10^{1}	1.73341×10^{1}	+	1.21393×10^{2}	2.06836×10^{1}	+	3.83014×10^{2}	1.94612×10^{1}	+	1.34424×10^{2}	3.51533×10^{1}	+
F9	3.69012×10^{2}	3.14426×10^{2}	+	4.87484×10^{2}	3.76369×10^{2}	+	9.32487×10^{2}	5.46491×10^{2}	+	1.34684×10^{4}	1.78493×10^{3}	+	2.26943×10^{3}	1.70892×10^{3}	+
F10	$\mathbf{2.12631\times10^{3}}$	4.53978×10^{2}	−	2.18719×10^{3}	5.32636×10^{2}	−	3.87564×10^{3}	5.19271×10^{2}	−	7.22006×10^{3}	$\mathbf{2.76241\times10^{2}}$	+	2.97355×10^{3}	5.24757×10^{2}	−
F11	1.11547×10^{2}	9.18567×10^{1}	+	5.44280×10^{1}	2.95628×10^{1}	+	4.64119×10^{2}	1.90268×10^{2}	+	5.95437×10^{3}	1.04564×10^{3}	+	4.21766×10^{3}	4.23432×10^{3}	+
F12	2.66163×10^{6}	2.10403×10^{6}	+	1.88872×10^{6}	1.22620×10^{6}	+	9.16086×10^{6}	5.04138×10^{6}	+	4.18520×10^{9}	1.52185×10^{9}	+	5.81408×10^{6}	3.70449×10^{6}	+
F13	2.25486×10^{4}	2.13234×10^{4}	+	1.63857×10^{4}	1.97165×10^{4}	−	5.32564×10^{5}	1.09916×10^{6}	+	1.82253×10^{9}	8.93901×10^{8}	+	1.72315×10^{5}	1.32650×10^{5}	+
F14	5.97926×10^{4}	6.26809×10^{4}	+	2.26614×10^{4}	1.81378×10^{4}	+	3.86067×10^{5}	2.78778×10^{5}	+	4.53933×10^{5}	2.16940×10^{5}	+	2.10371×10^{6}	2.01004×10^{6}	+
F15	8.40682×10^{3}	1.06880×10^{4}	+	1.04698×10^{4}	1.10547×10^{4}	+	3.40162×10^{4}	1.98425×10^{4}	+	1.74992×10^{8}	9.02648×10^{7}	+	5.76133×10^{4}	5.13274×10^{4}	+
F16	9.81277×10^{2}	2.85600×10^{2}	+	9.29310×10^{2}	2.74389×10^{2}	+	1.07380×10^{3}	2.86736×10^{2}	+	2.82399×10^{3}	$\mathbf{2.23344\times10^{2}}$	+	1.31716×10^{3}	3.89057×10^{2}	+
F17	3.93310×10^{2}	1.69161×10^{2}	+	4.37571×10^{2}	1.74304×10^{2}	+	4.93606×10^{2}	1.79695×10^{2}	+	1.31101×10^{3}	1.79622×10^{2}	+	6.82748×10^{2}	2.14499×10^{2}	+
F18	2.23285×10^{5}	2.14848×10^{5}	−	2.13639×10^{5}	2.49788×10^{5}	−	1.19970×10^{6}	1.19850×10^{6}	+	8.74549×10^{6}	3.72752×10^{6}	+	2.96959×10^{6}	3.86768×10^{6}	+
F19	1.18122×10^{4}	1.36936×10^{4}	+	1.24520×10^{4}	1.36118×10^{4}	+	6.06263×10^{4}	3.88753×10^{4}	+	2.41777×10^{8}	1.18581×10^{8}	+	2.63049×10^{4}	2.49442×10^{4}	+
F20	4.70096×10^{2}	1.70721×10^{2}	+	4.33864×10^{2}	1.56289×10^{2}	+	4.85809×10^{2}	1.38919×10^{2}	+	8.11218×10^{2}	$\mathbf{8.74384\times10^{1}}$	+	6.59229×10^{2}	2.46687×10^{2}	+
F21	2.78276×10^{2}	2.00238×10^{1}	+	2.85032×10^{2}	2.22164×10^{1}	+	3.17207×10^{2}	2.32498×10^{1}	+	5.79257×10^{2}	2.41389×10^{1}	+	3.38249×10^{2}	3.42450×10^{1}	+
F22	2.20143×10^{3}	1.10016×10^{3}	+	2.26235×10^{3}	1.08940×10^{3}	+	3.11283×10^{3}	1.78468×10^{3}	+	5.49399×10^{3}	6.25295×10^{2}	+	3.43165×10^{3}	1.23409×10^{3}	+
F23	4.24406×10^{2}	1.85916×10^{1}	+	4.37291×10^{2}	2.21506×10^{1}	+	4.82461×10^{2}	2.11504×10^{1}	+	9.29105×10^{2}	5.30891×10^{1}	+	4.92488×10^{2}	4.12799×10^{1}	+
F24	5.15008×10^{2}	2.64229×10^{1}	+	5.15028×10^{2}	2.77538×10^{1}	+	5.81074×10^{2}	3.79203×10^{1}	+	1.00616×10^{3}	5.74100×10^{1}	+	7.32847×10^{2}	8.81439×10^{1}	+
F25	3.94769×10^{2}	1.48753×10^{1}	+	3.91248×10^{2}	1.08868×10^{1}	+	4.65485×10^{2}	4.28382×10^{1}	+	3.38047×10^{3}	8.91537×10^{2}	+	4.05507×10^{2}	1.99133×10^{1}	+
F26	1.90069×10^{3}	3.48743×10^{2}	+	1.92513×10^{3}	5.36919×10^{2}	+	2.37805×10^{3}	5.22486×10^{2}	+	6.95485×10^{3}	4.36811×10^{2}	+	2.44597×10^{3}	1.22535×10^{3}	+
F27	5.25889×10^{2}	1.35166×10^{1}	+	5.27048×10^{2}	1.09487×10^{1}	+	5.45375×10^{2}	1.18367×10^{1}	+	1.04915×10^{3}	1.09692×10^{2}	+	5.57504×10^{2}	3.21169×10^{1}	+
F28	4.34933×10^{2}	2.83807×10^{1}	+	4.24011×10^{2}	$\mathbf{1.84665\times10^{1}}$	+	5.35599×10^{2}	4.35077×10^{1}	+	3.27045×10^{3}	5.85101×10^{2}	+	4.36374×10^{2}	2.52075×10^{1}	+
F29	8.14006×10^{2}	1.91105×10^{2}	+	8.00075×10^{2}	2.18017×10^{2}	+	9.37953×10^{2}	2.14164×10^{2}	+	2.63427×10^{3}	2.03907×10^{2}	+	1.02749×10^{3}	2.39429×10^{2}	+
F30	1.06098×10^{4}	4.93029×10^{3}	+	1.04119×10^{4}	1.06968×10^{4}	+	6.06206×10^{5}	6.25287×10^{5}	+	1.99803×10^{8}	1.04920×10^{8}	+	4.81986×10^{4}	1.64438×10^{5}	+
+/−/=	**28/2/0**			**27/3/0**			**29/1/0**			**30/0/0**			**29/1/0**		

Table 6 Experimental results of IGHS, LHS, IMGHS, ID-HS-LDD, and DMDS-HS in CEC2017, when D = 30.

Function	IGHS			LHS			IMGHS			ID-HS-LDD			DMDS-HS	
	Mean	Std	Sign	Mean	Std	Sign	Mean	Std	Sign	Mean	Std	Sign	Mean	Std
F1	5.65933×10^{9}	1.25429×10^{9}	+	4.69323×10^{3}	5.19265×10^{3}	+	6.60033×10^{3}	6.43017×10^{3}	+	5.70117×10^{3}	6.42193×10^{3}	+	3.91849×10^{3}	4.88540×10^{3}
F2	2.87098×10^{30}	1.31963×10^{31}	+	8.53329×10^{4}	5.67845×10^{5}	+	1.52004×10^{-3}	2.24541×10^{-3}	+	1.47661×10^{8}	1.05433×10^{9}	+	**3.14537×10^{-5}**	**2.31330×10^{-5}**
F3	2.87314×10^{4}	5.46918×10^{3}	+	3.18831×10^{3}	1.83315×10^{3}	+	7.07491×10^{2}	1.53312×10^{3}	+	9.64967×10^{3}	4.32992×10^{3}	+	**1.67371×10^{-7}**	**2.68714×10^{-8}**
F4	8.48530×10^{2}	1.79299×10^{2}	+	7.87745×10^{1}	3.03639×10^{1}	+	**2.16209×10^{1}**	**2.59581×10^{1}**	–	2.46201×10^{1}	3.07919×10^{1}	–	6.77635×10^{1}	3.06724×10^{1}
F5	2.62576×10^{2}	1.70614×10^{1}	+	1.00208×10^{2}	2.59164×10^{1}	+	1.51856×10^{2}	3.87064×10^{1}	+	2.18124×10^{2}	2.47057×10^{1}	+	**3.14796×10^{1}**	1.71481×10^{1}
F6	3.14626×10^{1}	4.93354×10	+	**3.96173×10^{-4}**	**8.71211×10^{-4}**	–	1.50803×10^{-1}	1.09652×10^{-1}	+	6.94835×10^{-3}	2.01696×10^{-2}	+	4.22318×10^{-4}	2.01487×10^{-3}
F7	3.60548×10^{2}	2.66345×10^{1}	+	1.32961×10^{2}	2.68566×10^{1}	+	2.08580×10^{2}	4.59914×10^{1}	+	2.52238×10^{2}	2.13927×10^{1}	+	**6.42530×10^{1}**	**1.26582×10^{1}**
F8	2.38310×10^{2}	1.47955×10^{1}	+	1.11415×10^{2}	2.73212×10^{1}	+	1.48290×10^{2}	3.32618×10^{1}	+	2.13276×10^{2}	1.95865×10^{1}	+	**3.24090×10^{1}**	**1.10774×10^{1}**
F9	3.30355×10^{3}	9.85819×10^{2}	+	8.24252×10^{2}	6.92514×10^{2}	+	3.14124×10^{3}	1.50095×10^{3}	+	4.17754×10^{1}	1.20721×10^{2}	+	**5.68414**	**6.53559**
F10	6.88871×10^{3}	3.03718×10^{2}	+	2.62890×10^{3}	5.10464×10^{2}	–	3.18001×10^{3}	5.21376×10^{2}	–	7.04624×10^{3}	3.04210×10^{2}	+	5.39402×10^{3}	1.61564×10^{3}
F11	4.14922×10^{2}	1.95281×10^{2}	+	6.90106×10^{1}	**2.85833×10^{1}**	+	9.62033×10^{1}	3.77620×10^{1}	+	7.80426×10^{1}	6.27776×10^{1}	+	**4.49341×10^{1}**	3.23084×10^{1}
F12	9.97533×10^{7}	4.60997×10^{7}	+	1.15058×10^{6}	1.10180×10^{6}	+	**4.82118×10^{4}**	**3.82172×10^{4}**	–	1.10820×10^{5}	7.95729×10^{4}	–	1.06557×10^{5}	6.36326×10^{4}
F13	9.45501×10^{5}	7.68482×10^{5}	+	1.61067×10^{4}	2.01457×10^{4}	–	2.23477×10^{4}	1.99075×10^{4}	+	**6.76382×10^{3}**	**8.14385×10^{3}**	–	1.77246×10^{4}	1.75158×10^{4}
F14	2.09156×10^{5}	2.32689×10^{5}	+	2.33929×10^{5}	1.52641×10^{5}	+	2.48508×10^{4}	1.66591×10^{4}	+	1.32130×10^{4}	1.41319×10^{4}	+	**3.79748×10^{3}**	**3.56795×10^{3}**
F15	1.32667×10^{5}	3.48084×10^{5}	+	5.43809×10^{3}	**5.61951×10^{3}**	+	7.52618×10^{3}	8.68117×10^{3}	+	**4.71845×10^{3}**	6.26198×10^{3}	–	7.08304×10^{3}	8.99773×10^{3}
F16	1.83138×10^{3}	2.42372×10^{2}	+	1.19531×10^{3}	3.20906×10^{2}	+	1.37788×10^{3}	3.11674×10^{2}	+	1.28396×10^{3}	5.81342×10^{2}	+	**4.01664×10^{2}**	3.26430×10^{2}
F17	3.16404×10^{2}	1.49633×10^{2}	+	4.75709×10^{2}	2.42404×10^{2}	+	7.04020×10^{2}	2.72375×10^{2}	+	1.58777×10^{2}	1.18283×10^{2}	+	**1.15589×10^{2}**	**9.46961×10^{1}**
F18	1.10441×10^{6}	1.03217×10^{6}	+	1.19499×10^{6}	1.24838×10^{6}	+	**1.33839×10^{5}**	**9.16169×10^{4}**	–	4.71492×10^{5}	3.53831×10^{5}	+	2.05959×10^{5}	1.65101×10^{5}
F19	1.87095×10^{5}	2.44307×10^{5}	+	8.19905×10^{3}	1.02856×10^{4}	–	9.78325×10^{3}	1.21482×10^{4}	+	**5.59102×10^{3}**	**6.79247×10^{3}**	–	8.47925×10^{3}	9.35787×10^{3}
F20	3.41497×10^{2}	1.33704×10^{2}	+	5.08102×10^{2}	2.40137×10^{2}	+	6.98522×10^{2}	2.15132×10^{2}	+	1.63397×10^{2}	1.11447×10^{2}	+	**1.25165×10^{2}**	1.06272×10^{2}
F21	4.43787×10^{2}	**1.81665×10^{1}**	+	3.05829×10^{2}	2.45330×10^{1}	+	3.51863×10^{2}	3.91137×10^{1}	+	4.04118×10^{2}	2.14195×10^{1}	+	**2.37888×10^{2}**	1.85514×10^{1}
F22	1.35580×10^{3}	1.72586×10^{3}	+	1.45149×10^{3}	1.68101×10^{3}	–	3.49960×10^{3}	1.24868×10^{3}	+	**1.00243×10^{2}**	**7.43704×10^{-1}**	–	2.53671×10^{3}	2.97997×10^{3}
F23	6.34314×10^{2}	2.34041×10^{1}	+	4.58013×10^{2}	2.56226×10^{1}	+	5.09632×10^{2}	4.21873×10^{1}	+	5.27699×10^{2}	6.41196×10^{1}	+	**3.84657×10^{2}**	**1.08045×10^{1}**
F24	7.01684×10^{2}	**1.97305×10^{1}**	+	7.06254×10^{2}	8.96803×10^{1}	+	7.78615×10^{2}	1.24893×10^{2}	+	6.21966×10^{2}	3.50108×10^{1}	+	**5.01346×10^{2}**	4.49124×10^{1}
F25	6.42908×10^{2}	5.86937×10^{1}	+	4.00413×10^{2}	2.31494×10^{1}	+	4.00170×10^{2}	1.80606×10^{1}	+	3.87401×10^{2}	5.84859×10	–	**3.87106×10^{2}**	**7.82030×10^{-1}**
F26	3.51940×10^{3}	9.03493×10^{2}	+	2.02427×10^{3}	1.18385×10^{3}	+	2.80911×10^{3}	1.19009×10^{3}	+	**1.28982×10^{3}**	1.13801×10^{3}	–	1.37412×10^{3}	**9.24367×10^{1}**
F27	6.04677×10^{2}	3.02879×10^{1}	+	5.42806×10^{2}	1.98666×10^{1}	+	5.71707×10^{2}	2.82078×10^{1}	+	5.09992×10^{2}	1.13077×10^{1}	+	**5.05911×10^{2}**	**6.10333**
F28	7.65456×10^{2}	9.46092×10^{1}	+	3.68475×10^{2}	5.62002×10^{1}	–	3.48044×10^{2}	6.14817×10^{1}	+	**3.33243×10^{2}**	5.27641×10^{1}	–	3.69846×10^{2}	7.02718×10^{1}
F29	1.31625×10^{3}	2.68595×10^{2}	+	9.20382×10^{2}	2.51686×10^{2}	+	1.03511×10^{3}	2.44088×10^{2}	+	5.86124×10^{2}	1.39216×10^{2}	+	**5.18119×10^{2}**	**1.15098×10^{2}**
F30	2.80243×10^{6}	4.45310×10^{6}	+	6.69605×10^{3}	4.00902×10^{3}	–	9.04043×10^{3}	4.51132×10^{3}	+	**4.83946×10^{3}**	**2.51407×10^{3}**	–	7.48488×10^{3}	3.48633×10^{3}
+/−/=	30/0/0			23/7/0			26/4/0			20/10/0			—	

Table 7 Experimental results of HS, IHS, GHS, SGHS, and NGHS in CEC2017, when D = 50.

Function	HS			IHS			GHS			SGHS			NGHS		
	Mean	Std	Sign	Mean	Std	Sign	Mean	Std	Sign	Mean	Std	Sign	Mean	Std	Sign
F1	1.59045×10^{7}	5.78454×10^{6}	+	3.90710×10^{4}	1.69441×10^{5}	+	4.80559×10^{9}	8.70276×10^{8}	+	1.15508×10^{11}	2.12205×10^{10}	+	2.69991×10^{5}	1.28393×10^{5}	+
F2	1.62924×10^{40}	1.16169×10^{41}	+	3.39750×10^{33}	1.96842×10^{34}	+	1.79940×10^{56}	1.26151×10^{57}	+	4.72599×10^{73}	1.10108×10^{74}	+	1.28873×10^{50}	9.20338×10^{50}	+
F3	2.27081×10^{4}	4.37891×10^{3}	+	2.27420×10^{4}	4.96451×10^{3}	+	1.01042×10^{5}	1.45668×10^{4}	+	2.26608×10^{5}	2.07480×10^{4}	+	1.66660×10^{5}	4.63488×10^{4}	+
F4	2.14009×10^{2}	5.30056×10^{1}	+	1.95328×10^{2}	5.28257×10^{1}	+	7.34538×10^{2}	1.44329×10^{2}	+	2.76384×10^{4}	5.97805×10^{3}	+	1.43215×10^{2}	5.00468×10^{1}	+
F5	1.43366×10^{2}	2.55159×10^{1}	+	1.43207×10^{2}	**2.48426×10^{1}**	+	3.39087×10^{2}	3.71957×10^{1}	+	8.26142×10^{2}	3.77657×10^{1}	+	2.26614×10^{2}	4.60295×10^{1}	+
F6	1.14264	2.71236×10^{-1}	+	1.24240	3.57004×10^{-1}	+	1.47675×10^{1}	2.19452	+	1.02723×10^{2}	6.47054	+	5.35188×10^{-2}	3.71110×10^{-2}	+
F7	2.65051×10^{2}	3.10820×10^{1}	+	2.62912×10^{2}	4.73501×10^{1}	+	5.63904×10^{2}	4.28653×10^{1}	+	2.96566×10^{3}	3.31861×10^{2}	+	2.72321×10^{2}	4.71220×10^{1}	+
F8	1.38037×10^{2}	2.44490×10^{1}	+	1.46310×10^{2}	2.77986×10^{1}	+	3.40925×10^{2}	3.11518×10^{1}	+	8.21273×10^{2}	4.10631×10^{1}	+	2.43781×10^{2}	4.97421×10^{1}	+
F9	1.41684×10^{3}	8.49555×10^{2}	+	1.75350×10^{3}	9.55670×10^{2}	+	6.31607×10^{3}	2.44869×10^{3}	+	4.62683×10^{4}	3.96940×10^{3}	+	5.33242×10^{3}	2.67424×10^{3}	+
F10	**4.13754×10^{3}**	5.98015×10^{2}	−	4.19590×10^{3}	6.59605×10^{2}	−	9.63082×10^{3}	8.38273×10^{2}	−	1.34265×10^{4}	4.01744×10^{2}	+	4.97307×10^{3}	6.29454×10^{2}	−
F11	4.50487×10^{2}	2.75050×10^{2}	+	2.05744×10^{2}	1.67646×10^{2}	+	1.48682×10^{3}	6.76409×10^{2}	+	2.03990×10^{4}	3.41739×10^{3}	+	6.78392×10^{3}	5.41659×10^{3}	+
F12	1.05957×10^{7}	5.83790×10^{6}	+	1.06530×10^{7}	6.06777×10^{6}	+	2.35180×10^{8}	8.73633×10^{7}	+	3.42509×10^{10}	1.00592×10^{10}	+	1.18836×10^{7}	6.90579×10^{6}	+
F13	9.12638×10^{3}	9.94214×10^{3}	−	8.13150×10^{3}	8.32300×10^{3}	−	1.52536×10^{6}	7.24643×10^{5}	+	1.08594×10^{10}	3.97155×10^{9}	+	1.75146×10^{5}	2.33977×10^{5}	+
F14	1.92447×10^{5}	1.37531×10^{5}	+	6.99240×10^{4}	4.56810×10^{4}	+	2.72063×10^{6}	1.69212×10^{6}	+	9.19466×10^{6}	3.02478×10^{6}	+	6.79325×10^{6}	4.47006×10^{6}	+
F15	9.30384×10^{3}	6.96438×10^{3}	+	7.39700×10^{3}	6.80748×10^{3}	+	1.29862×10^{5}	8.87402×10^{4}	+	3.01068×10^{9}	9.97330×10^{8}	+	8.93051×10^{4}	6.59754×10^{4}	+
F16	1.78305×10^{3}	4.17256×10^{2}	+	1.66690×10^{3}	3.88409×10^{2}	+	2.07013×10^{3}	4.14706×10^{2}	+	5.50500×10^{3}	4.39483×10^{2}	+	2.26289×10^{3}	4.35669×10^{2}	+
F17	1.18055×10^{3}	3.40604×10^{2}	+	1.21020×10^{3}	2.96478×10^{2}	+	1.42005×10^{3}	3.07845×10^{2}	+	4.93862×10^{3}	5.75108×10^{2}	+	1.63094×10^{3}	3.48044×10^{2}	+
F18	2.43998×10^{6}	1.92417×10^{6}	+	9.01100×10^{5}	8.46155×10^{5}	+	1.08834×10^{7}	7.56549×10^{6}	+	4.58122×10^{7}	1.49863×10^{7}	+	9.81107×10^{6}	8.76999×10^{6}	+
F19	1.68957×10^{4}	1.31100×10^{4}	+	1.63040×10^{4}	1.28062×10^{4}	+	8.57369×10^{4}	4.34604×10^{4}	+	9.92663×10^{8}	4.97089×10^{8}	+	2.15153×10^{4}	1.60724×10^{4}	+
F20	1.03609×10^{3}	2.96372×10^{2}	+	1.05600×10^{3}	2.89379×10^{2}	+	1.13986×10^{3}	2.98138×10^{2}	+	1.97725×10^{3}	**1.74595×10^{2}**	+	1.28258×10^{3}	3.63733×10^{2}	+
F21	3.42602×10^{2}	2.42151×10^{1}	+	3.38571×10^{2}	2.52459×10^{1}	+	5.46114×10^{2}	3.71151×10^{1}	+	1.02887×10^{3}	3.81728×10^{1}	+	4.60025×10^{2}	5.07663×10^{1}	+
F22	4.99397×10^{3}	6.62310×10^{2}	−	4.96380×10^{3}	7.41662×10^{2}	−	1.04507×10^{4}	8.83233×10^{2}	−	1.38956×10^{4}	**3.74672×10^{2}**	+	6.11891×10^{3}	7.04308×10^{2}	−
F23	5.90659×10^{2}	3.20815×10^{1}	+	5.98369×10^{2}	4.17007×10^{1}	+	8.11877×10^{2}	3.81306×10^{1}	+	1.65851×10^{3}	9.51766×10^{1}	+	6.99673×10^{2}	4.56673×10^{1}	+
F24	6.52450×10^{2}	6.17300×10^{1}	−	6.30931×10^{2}	4.99611×10^{1}	−	9.67878×10^{2}	**2.57079×10^{1}**	+	1.79019×10^{3}	1.21487×10^{2}	+	1.23501×10^{3}	1.99136×10^{2}	+
F25	5.96323×10^{2}	2.88130×10^{1}	+	5.84007×10^{2}	3.18604×10^{1}	+	1.01225×10^{3}	1.32082×10^{2}	+	1.67109×10^{4}	3.78059×10^{3}	+	5.81601×10^{2}	**2.58749×10^{1}**	+
F26	2.84802×10^{3}	4.75812×10^{2}	+	2.97680×10^{3}	3.22092×10^{2}	+	5.08230×10^{3}	3.27161×10^{2}	+	1.44913×10^{4}	1.18662×10^{3}	+	3.90478×10^{3}	1.01987×10^{3}	+
F27	6.70131×10^{2}	7.01822×10^{1}	+	6.85819×10^{2}	8.42710×10^{1}	+	8.65843×10^{2}	7.57981×10^{1}	+	2.62531×10^{3}	2.15545×10^{2}	+	9.06200×10^{2}	9.98768×10^{1}	+
F28	5.60484×10^{2}	4.19526×10^{1}	+	5.52112×10^{2}	4.06194×10^{1}	+	1.05293×10^{3}	1.57059×10^{2}	+	9.42882×10^{3}	1.44083×10^{3}	+	5.33077×10^{2}	2.61533×10^{1}	+
F29	1.04599×10^{3}	2.59659×10^{2}	+	1.03380×10^{3}	2.58412×10^{2}	+	1.56895×10^{3}	3.39320×10^{2}	+	8.42860×10^{3}	1.59384×10^{3}	+	1.56628×10^{3}	3.60328×10^{2}	+
F30	1.19322×10^{6}	4.06663×10^{5}	+	1.19100×10^{6}	3.44676×10^{5}	+	6.76439×10^{6}	2.26086×10^{6}	+	2.32244×10^{9}	6.69656×10^{8}	+	8.13244×10^{6}	5.00114×10^{7}	+
+/−/=	**26/4/0**			**26/4/0**			**28/2/0**			**30/0/0**			**28/2/0**		

Table 8 Experimental results of IGHS, LHS, IMGHS, ID-HS-LDD, and DMDS-HS in CEC2017, when D = 50.

Function	IGHS			LHS			IMGHS			ID-HS-LDD			DMDS-HS	
	Mean	Std	Sign	Mean	Std	Sign	Mean	Std	Sign	Mean	Std	Sign	Mean	Std
F1	3.19607×10^{10}	3.50942×10^{9}	+	**4.19431×10^{3}**	**5.61831×10^{3}**	−	7.45128×10^{3}	7.31166×10^{3}	−	6.19374×10^{3}	7.09899×10^{3}	−	1.08742×10^{4}	1.00362×10^{4}
F2	1.46452×10^{62}	5.96358×10^{62}	+	$3.92561\times10+^{10}$	1.99596×10^{11}	+	**3.77895×10^{-4}**	**2.77267×10^{-4}**	+	8.34987×10^{44}	4.14909×10^{45}	+	1.37682	9.83215
F3	9.42338×10^{4}	1.19486×10^{4}	+	1.28852×10^{4}	6.85591×10^{3}	+	1.31110×10^{3}	1.76665×10^{3}	+	6.92044×10^{4}	1.22931×10^{4}	+	**5.27035×10^{2}**	**8.45589×10^{2}**
F4	4.92072×10^{3}	7.49549×10^{2}	+	1.11524×10^{2}	4.88745×10^{1}	+	**3.36961×10^{1}**	**3.69716×10^{1}**	−	9.07506×10^{1}	5.02965×10^{1}	+	8.12606×10^{1}	5.40156×10^{1}
F5	5.33225×10^{2}	2.54546×10^{1}	+	1.95216×10^{2}	3.48168×10^{1}	+	2.63265×10^{2}	5.33344×10^{1}	+	4.42774×10^{2}	2.91076×10^{1}	+	**7.26749×10^{1}**	2.73674×10^{1}
F6	5.63213×10^{1}	4.24886	+	**6.49313×10^{-4}**	**9.73608×10^{-4}**	−	1.31258×10^{-1}	6.44757×10^{-2}	+	1.86807	2.24412	+	3.09229×10^{-3}	5.28307×10^{-3}
F7	7.98328×10^{2}	4.18679×10^{1}	+	2.55900×10^{2}	4.10521×10^{1}	+	3.83545×10^{2}	5.77590×10^{1}	+	5.32505×10^{2}	3.73421×10^{1}	+	**1.20571×10^{2}**	**1.35692×10^{1}**
F8	5.26335×10^{2}	**2.33663×10^{1}**	+	1.95877×10^{2}	3.33884×10^{1}	+	2.62063×10^{2}	5.42656×10^{1}	+	4.43063×10^{2}	3.34348×10^{1}	+	**6.94982×10^{1}**	2.50691×10^{1}
F9	1.82283×10^{4}	3.08777×10^{3}	+	3.14759×10^{3}	1.85079×10^{3}	+	8.44862×10^{3}	3.75350×10^{3}	+	3.78503×10^{3}	3.18205×10^{3}	+	**4.15470×10^{1}**	**2.90951×10^{1}**
F10	1.31224×10^{4}	3.60416×10^{2}	+	4.41236×10^{3}	6.63686×10^{2}	−	5.28910×10^{3}	6.56068×10^{2}	−	1.32132×10^{4}	**3.29452×10^{2}**	+	1.15721×10^{4}	1.99547×10^{3}
F11	2.11340×10^{3}	5.96684×10^{2}	+	1.88772×10^{2}	8.38270×10^{1}	+	1.67480×10^{2}	4.88067×10^{1}	+	3.53299×10^{2}	9.84806×10^{1}	+	**6.10261×10^{1}**	**3.32062×10^{1}**
F12	3.27106×10^{9}	6.25551×10^{8}	+	2.92384×10^{6}	1.48033×10^{6}	+	**2.43764×10^{5}**	**1.46955×10^{5}**	−	8.32660×10^{5}	4.31704×10^{5}	−	1.34499×10^{6}	8.80787×10^{5}
F13	1.60552×10^{7}	1.08036×10^{7}	+	**5.64377×10^{3}**	7.31872×10^{3}	−	9.07875×10^{3}	9.40219×10^{3}	−	5.73398×10^{3}	**5.92067×10^{3}**	−	8.02957×10^{3}	7.46898×10^{3}
F14	1.24245×10^{6}	8.26136×10^{5}	+	3.63465×10^{5}	2.31019×10^{5}	+	3.34724×10^{4}	2.04486×10^{4}	+	4.88950×10^{4}	2.92906×10^{4}	+	**2.01301×10^{4}**	**1.77818×10^{4}**
F15	3.90557×10^{5}	5.52234×10^{5}	+	7.63902×10^{3}	6.56048×10^{3}	+	1.11009×10^{4}	6.98060×10^{3}	+	**2.95662×10^{3}**	**3.58606×10^{3}**	−	5.23424×10^{3}	6.39444×10^{3}
F16	3.62105×10^{3}	**3.10637×10^{2}**	+	2.12087×10^{3}	4.70024×10^{2}	+	2.19453×10^{3}	3.95284×10^{2}	+	3.16217×10^{3}	4.88031×10^{2}	+	**1.42196×10^{3}**	6.91785×10^{2}
F17	2.18305×10^{3}	**1.88132×10^{2}**	+	1.41225×10^{3}	3.49631×10^{2}	+	1.69899×10^{3}	3.68795×10^{2}	+	1.87282×10^{3}	2.92798×10^{2}	+	**7.68211×10^{2}**	4.02005×10^{2}
F18	5.76066×10^{6}	4.06822×10^{6}	+	1.21545×10^{6}	8.11542×10^{5}	+	1.47591×10^{5}	**7.09061×10^{4}**	+	1.69531×10^{6}	1.29412×10^{6}	+	**1.45167×10^{5}**	1.19744×10^{5}
F19	2.51044×10^{5}	5.61414×10^{5}	+	1.87331×10^{4}	1.11130×10^{4}	+	1.97387×10^{4}	1.30245×10^{4}	+	**1.12156×10^{4}**	**8.29270×10^{3}**	+	1.25630×10^{4}	1.15351×10^{4}
F20	1.55043×10^{3}	3.25375×10^{2}	+	1.14310×10^{3}	2.81672×10^{2}	+	1.33005×10^{3}	3.33224×10^{2}	+	1.68731×10^{3}	2.03467×10^{2}	+	**9.53178×10^{2}**	4.70142×10^{2}
F21	7.35804×10^{2}	2.82926×10^{1}	+	4.10420×10^{2}	4.46436×10^{1}	+	4.78619×10^{2}	4.86691×10^{1}	+	6.40929×10^{2}	**2.39943×10^{1}**	+	**2.71860×10^{2}**	3.47111×10^{1}
F22	1.21915×10^{4}	3.24909×10^{3}	+	5.28395×10^{3}	1.03177×10^{3}	−	6.42426×10^{3}	7.67218×10^{2}	−	1.12162×10^{4}	5.20722×10^{3}	+	1.14905×10^{4}	2.21895×10^{3}
F23	1.09860×10^{3}	3.84004×10^{1}	+	6.78871×10^{2}	5.31366×10^{1}	+	7.54139×10^{2}	7.19590×10^{1}	+	8.87474×10^{2}	4.38206×10^{1}	+	**5.02695×10^{2}**	**1.80407×10^{1}**
F24	1.14150×10^{3}	4.49944×10^{1}	+	1.12579×10^{3}	1.48145×10^{2}	+	1.22407×10^{3}	1.84331×10^{2}	+	9.64258×10^{2}	2.90034×10^{1}	+	6.74187×10^{2}	1.02697×10^{2}
F25	3.32300×10^{3}	4.49302×10^{2}	+	5.59211×10^{2}	2.99739×10^{1}	+	5.58496×10^{2}	4.13231×10^{1}	+	**5.16774×10^{2}**	3.76675×10^{1}	−	5.19922×10^{2}	3.92342×10^{1}
F26	7.61651×10^{3}	7.88865×10^{2}	+	3.63882×10^{3}	1.39324×10^{3}	+	4.71319×10^{3}	1.14487×10^{3}	+	5.12899×10^{3}	1.63739×10^{3}	+	**1.94516×10^{3}**	**1.60184×10^{2}**
F27	1.38475×10^{3}	1.11105×10^{2}	+	7.88887×10^{2}	9.51233×10^{1}	+	9.65783×10^{2}	1.38753×10^{2}	+	5.74478×10^{2}	5.83911×10^{1}	+	**5.60891×10^{2}**	**2.83407×10^{1}**
F28	3.03322×10^{3}	3.18436×10^{2}	+	5.03351×10^{2}	2.48580×10^{1}	+	4.94430×10^{2}	3.16415×10^{1}	+	4.80490×10^{2}	**2.25731×10^{1}**	+	**4.77455×10^{2}**	2.55058×10^{1}
F29	3.34482×10^{3}	3.09926×10^{2}	+	1.27434×10^{3}	3.50994×10^{2}	+	1.55771×10^{3}	3.58258×10^{2}	+	1.05531×10^{3}	5.95177×10^{2}	+	**5.20648×10^{2}**	**1.28560×10^{2}**
F30	5.05897×10^{7}	1.82220×10^{7}	+	9.87257×10^{5}	2.64394×10^{5}	+	9.89557×10^{5}	1.84471×10^{5}	+	9.58279×10^{5}	**1.50992×10^{5}**	+	**9.06384×10^{5}**	1.81187×10^{5}
+/−/=	30/0/0			25/5/0			24/6/0			25/5/0			—	

Table 9 Results obtained from 51 independent runs of DMDS-HS and four other advanced algorithms on the 30*D* benchmark of CEC2017.

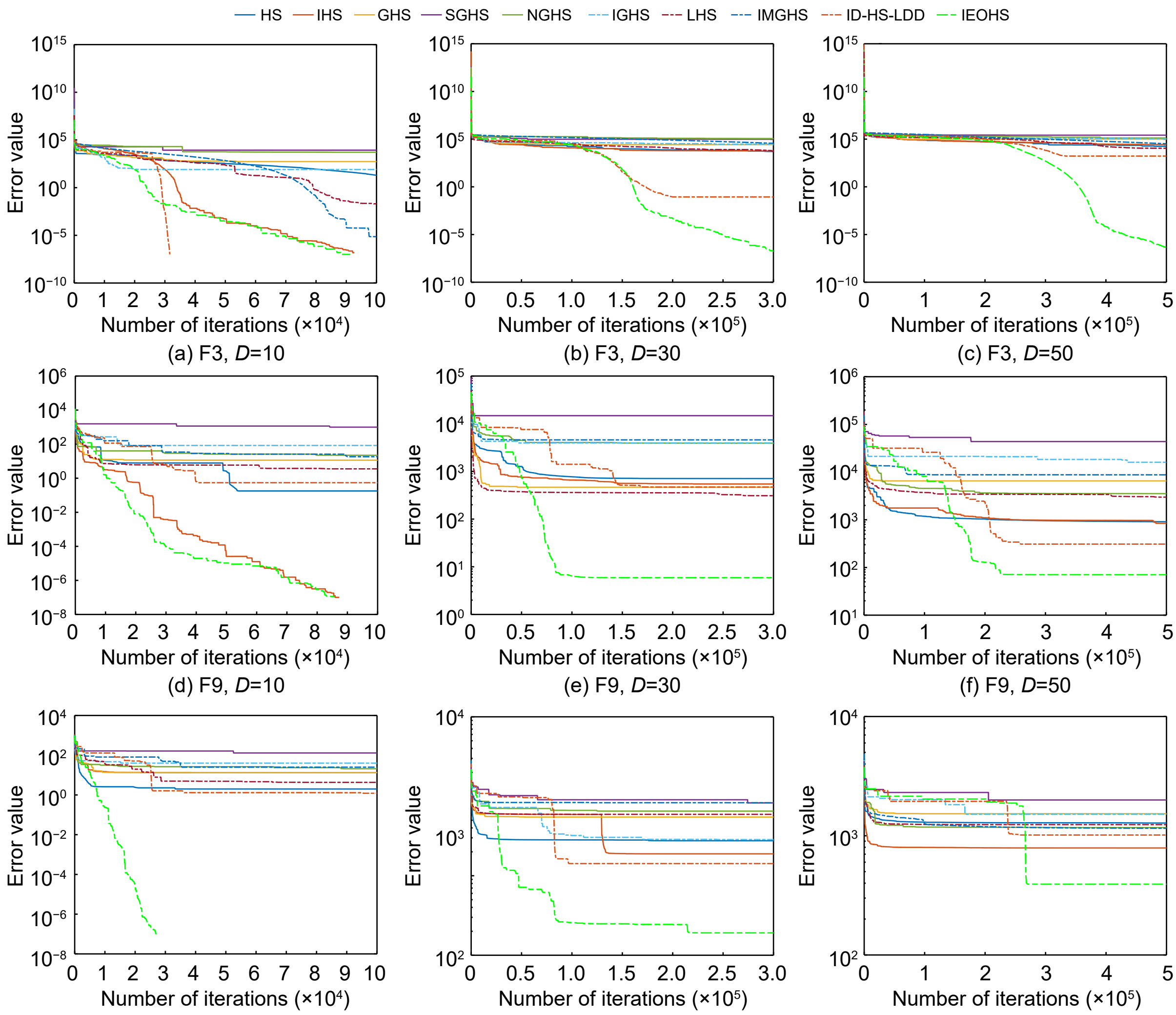
Function	Statistical index	SLWCHOA	IWOA	HGWO	GWO	DMDS-HS
F1	Mean	3.1427025×10^{10}	1.2208871×10^{9}	4.8193326×10^{10}	1.4202351×10^{9}	$\mathbf{3.9184942\times10^{3}}$
	Std	4.7994355×10^{9}	3.6892701×10^{8}	8.5282402×10^{9}	1.1333496×10^{9}	$\mathbf{4.8853985\times10^{3}}$
F2	Mean	1.5959448×10^{34}	1.1194591×10^{30}	1.0428313×10^{48}	2.3557110×10^{30}	$\mathbf{3.1453736\times10^{-5}}$
	Std	1.7595367×10^{34}	3.3549497×10^{30}	3.4984481×10^{48}	1.5939404×10^{31}	$\mathbf{2.3132986\times10^{-5}}$
F3	Mean	6.2389399×10^{4}	3.9769285×10^{4}	9.1600059×10^{4}	3.2412839×10^{4}	$\mathbf{1.6737079\times10^{-7}}$
	Std	6.9179740×10^{3}	7.6071728×10^{3}	3.5042174×10^{3}	1.1585595×10^{4}	$\mathbf{2.6871356\times10^{-8}}$
F4	Mean	3.8680228×10^{3}	3.3645362×10^{2}	1.3516019×10^{4}	1.6886969×10^{2}	$\mathbf{6.7763496\times10^{1}}$
	Std	1.2172326×10^{3}	9.1809914×10^{1}	3.1210514×10^{3}	6.2175849×10^{1}	$\mathbf{3.0672450\times10^{1}}$
F5	Mean	3.5199300×10^{2}	2.4577646×10^{2}	4.1224161×10^{2}	9.3997063×10^{1}	$\mathbf{3.1479577\times10^{1}}$
	Std	2.0713916×10^{1}	3.7244508×10^{1}	3.2831866×10^{1}	2.5007342×10^{1}	$\mathbf{1.7148106\times10^{1}}$
F6	Mean	7.6311156×10^{1}	5.9059948×10^{1}	9.4220202×10^{1}	6.7980890	$\mathbf{4.2231825\times10^{-4}}$
	Std	6.9288576	6.4927645	7.5657627	3.9941614	$\mathbf{2.0148732\times10^{-3}}$
F7	Mean	5.7007631×10^{2}	4.5336113×10^{2}	6.9383060×10^{2}	1.5745060×10^{2}	$\mathbf{6.4253003\times10^{1}}$
	Std	3.7220648×10^{1}	7.1421021×10^{1}	4.5828077×10^{1}	4.4862857×10^{1}	$\mathbf{1.2658195\times10^{1}}$
F8	Mean	2.8542480×10^{2}	1.7289221×10^{2}	3.5266079×10^{2}	8.2155180×10^{1}	$\mathbf{3.2408977\times10^{1}}$
	Std	2.6268993×10^{1}	2.5334267×10^{1}	2.4635760×10^{1}	2.0053676×10^{1}	$\mathbf{1.1077443\times10^{1}}$
F9	Mean	7.7736968×10^{3}	4.6051388×10^{3}	1.0473333×10^{4}	6.4525544×10^{2}	**5.6841394**
	Std	1.4670260×10^{3}	8.7373344×10^{2}	1.6559261×10^{3}	3.8385034×10^{2}	**6.5355946**
F10	Mean	7.1584675×10^{3}	4.4124286×10^{3}	8.7337780×10^{3}	$\mathbf{2.9946683\times10^{3}}$	5.3940217×10^{3}
	Std	$\mathbf{2.8960499\times10^{2}}$	6.0964539×10^{2}	4.2458637×10^{2}	4.9432462×10^{2}	1.6156362×10^{3}
F11	Mean	2.8518966×10^{3}	5.2334006×10^{2}	9.2231107×10^{3}	6.7477169×10^{2}	$\mathbf{4.4934135\times10^{1}}$
	Std	7.3025243×10^{2}	1.6564709×10^{2}	2.3561085×10^{3}	7.7363736×10^{2}	$\mathbf{3.2308424\times10^{1}}$
F12	Mean	6.0505633×10^{9}	2.0066875×10^{8}	1.0590692×10^{10}	4.3195995×10^{7}	$\mathbf{1.0655696\times10^{5}}$
	Std	2.0502741×10^{9}	1.5212784×10^{8}	3.1097009×10^{9}	4.6653248×10^{7}	$\mathbf{6.3632639\times10^{4}}$
F13	Mean	2.0864503×10^{9}	7.0339174×10^{5}	9.6721921×10^{9}	4.8399024×10^{6}	$\mathbf{1.7724568\times10^{4}}$
	Std	2.1668203×10^{9}	1.3302038×10^{6}	3.3462747×10^{9}	2.3289275×10^{7}	$\mathbf{1.7515804\times10^{4}}$
F14	Mean	5.8228236×10^{5}	3.7186265×10^{5}	7.7933310×10^{6}	2.4391080×10^{5}	$\mathbf{3.7974768\times10^{3}}$
	Std	5.4358562×10^{5}	3.4552073×10^{5}	5.8413120×10^{6}	3.6907518×10^{5}	$\mathbf{3.5679486\times10^{3}}$
F15	Mean	1.4252304×10^{7}	3.9664398×10^{4}	4.0143753×10^{8}	1.2494448×10^{6}	$\mathbf{7.0830400\times10^{3}}$
	Std	1.2761353×10^{7}	3.1342264×10^{4}	2.4168046×10^{8}	8.3678736×10^{6}	$\mathbf{8.9977284\times10^{3}}$
F16	Mean	2.3968770×10^{3}	1.7841680×10^{3}	4.3868527×10^{3}	7.3797149×10^{2}	$\mathbf{4.0166390\times10^{2}}$
	Std	3.6968330×10^{2}	3.8420053×10^{2}	7.6012752×10^{2}	$\mathbf{2.7827157\times10^{2}}$	3.2643004×10^{2}
F17	Mean	9.3463949×10^{2}	6.8668510×10^{2}	2.3789611×10^{3}	2.7914155×10^{2}	$\mathbf{1.1558930\times10^{2}}$
	Std	1.4554396×10^{2}	2.2978945×10^{2}	8.6254730×10^{2}	1.3737704×10^{2}	$\mathbf{9.4696070\times10^{1}}$
F18	Mean	2.1678066×10^{6}	1.6633286×10^{6}	7.7890397×10^{7}	6.2139461×10^{5}	$\mathbf{2.0595874\times10^{5}}$
	Std	1.3324311×10^{6}	2.5533463×10^{6}	5.2245843×10^{7}	1.0506143×10^{6}	$\mathbf{1.6510099\times10^{5}}$
F19	Mean	2.0193291×10^{8}	2.0513030×10^{6}	7.2649748×10^{8}	1.2380925×10^{6}	$\mathbf{8.4792548\times10^{3}}$
	Std	1.9889231×10^{8}	1.7144994×10^{6}	3.9535317×10^{8}	4.6475474×10^{6}	$\mathbf{9.3578668\times10^{3}}$
F20	Mean	8.2824827×10^{2}	5.8372885×10^{2}	1.3246706×10^{3}	3.3178584×10^{2}	$\mathbf{1.2516535\times10^{2}}$
	Std	1.4808274×10^{2}	1.4955538×10^{2}	2.1727324×10^{2}	1.2323846×10^{2}	$\mathbf{1.0627209\times10^{2}}$
F21	Mean	5.1959229×10^{2}	4.2863287×10^{2}	6.3654587×10^{2}	2.8255931×10^{2}	$\mathbf{2.3788755\times10^{2}}$
	Std	3.1170052×10^{1}	4.0452815×10^{1}	4.4094083×10^{1}	2.8178945×10^{1}	$\mathbf{1.8551435\times10^{1}}$
F22	Mean	7.0679532×10^{3}	2.5472169×10^{3}	8.3158280×10^{3}	$\mathbf{1.9362961\times10^{3}}$	2.5367093×10^{3}
	Std	$\mathbf{4.5043219\times10^{2}}$	2.1257057×10^{3}	9.1587382×10^{2}	1.5454537×10^{3}	2.9799674×10^{3}
F23	Mean	7.6606609×10^{2}	6.9099253×10^{2}	1.3958328×10^{3}	4.4675624×10^{2}	$\mathbf{3.8465750\times10^{2}}$
	Std	4.0337742×10^{1}	5.9637639×10^{1}	1.7299173×10^{2}	3.6840099×10^{1}	$\mathbf{1.0804514\times10^{1}}$

(to be continued)

Table 9 Results obtained from 51 independent runs of DMDS-HS and four other advanced algorithms on the 30*D* benchmark of CEC2017.

(continued)

Function	Statistical index	SLWCHOA	IWOA	HGWO	GWO	DMDS-HS
F24	Mean	$8.668\,211\,8\times10^{2}$	$7.341\,208\,7\times10^{2}$	$1.560\,404\,4\times10^{3}$	$5.197\,010\,7\times10^{2}$	$\mathbf{5.013\,459\,5\times10^{2}}$
	Std	$\mathbf{3.993\,627\,4\times10^{1}}$	$6.263\,361\,1\times10^{1}$	$1.654\,725\,0\times10^{2}$	$4.765\,645\,9\times10^{1}$	$4.491\,241\,1\times10^{1}$
F25	Mean	$2.271\,138\,7\times10^{3}$	$5.314\,022\,4\times10^{2}$	$2.757\,212\,4\times10^{3}$	$4.696\,532\,5\times10^{2}$	$\mathbf{3.871\,056\,2\times10^{2}}$
	Std	$3.460\,778\,3\times10^{2}$	$3.247\,914\,7\times10^{1}$	$5.767\,903\,9\times10^{2}$	$3.576\,985\,4\times10^{1}$	$\mathbf{7.820\,303\,4\times10^{-1}}$
F26	Mean	$4.536\,702\,9\times10^{3}$	$4.634\,485\,5\times10^{3}$	$8.397\,731\,7\times10^{3}$	$1.980\,549\,8\times10^{3}$	$\mathbf{1.374\,117\,8\times10^{3}}$
	Std	$3.586\,373\,4\times10^{2}$	$1.240\,003\,0\times10^{3}$	$6.615\,004\,2\times10^{2}$	$3.573\,793\,9\times10^{2}$	$\mathbf{9.243\,666\,8\times10^{1}}$
F27	Mean	$9.019\,779\,7\times10^{2}$	$6.582\,015\,6\times10^{2}$	$2.227\,130\,9\times10^{3}$	$5.382\,939\,8\times10^{2}$	$\mathbf{5.059\,106\,2\times10^{2}}$
	Std	$7.579\,209\,8\times10^{1}$	$7.711\,016\,8\times10^{1}$	$4.656\,761\,1\times10^{2}$	$1.622\,795\,9\times10^{1}$	$\mathbf{6.103\,326\,4}$
F28	Mean	$2.258\,261\,1\times10^{3}$	$6.255\,926\,9\times10^{2}$	$4.074\,446\,7\times10^{3}$	$5.905\,632\,3\times10^{2}$	$\mathbf{3.698\,455\,4\times10^{2}}$
	Std	$6.278\,984\,7\times10^{2}$	$5.980\,354\,2\times10^{1}$	$6.974\,588\,3\times10^{2}$	$\mathbf{5.834\,748\,9\times10^{1}}$	$7.027\,179\,5\times10^{1}$
F29	Mean	$1.741\,740\,1\times10^{3}$	$1.797\,932\,8\times10^{3}$	$4.644\,595\,5\times10^{3}$	$7.986\,317\,4\times10^{2}$	$\mathbf{5.181\,186\,6\times10^{2}}$
	Std	$2.263\,840\,2\times10^{2}$	$2.697\,378\,2\times10^{2}$	$1.040\,186\,1\times10^{3}$	$1.621\,681\,1\times10^{2}$	$\mathbf{1.150\,981\,6\times10^{2}}$
F30	Mean	$7.043\,228\,9\times10^{7}$	$2.783\,026\,1\times10^{7}$	$1.266\,627\,4\times10^{9}$	$5.617\,874\,8\times10^{6}$	$\mathbf{7.484\,882\,3\times10^{3}}$
	Std	$2.220\,866\,2\times10^{7}$	$2.859\,018\,8\times10^{7}$	$6.605\,185\,0\times10^{8}$	$5.115\,620\,6\times10^{6}$	$\mathbf{3.486\,333\,1\times10^{3}}$
+/−/=		30/0/0	29/1/0	30/0/0	29/1/0	—



(a) F3, *D*=10 (b) F3, *D*=30 (c) F3, *D*=50

(d) F9, *D*=10 (e) F9, *D*=30 (f) F9, *D*=50

(g) F20, *D*=10 (h) F20, *D*=30 (i) F20, *D*=50

(to be continued)

(continued)

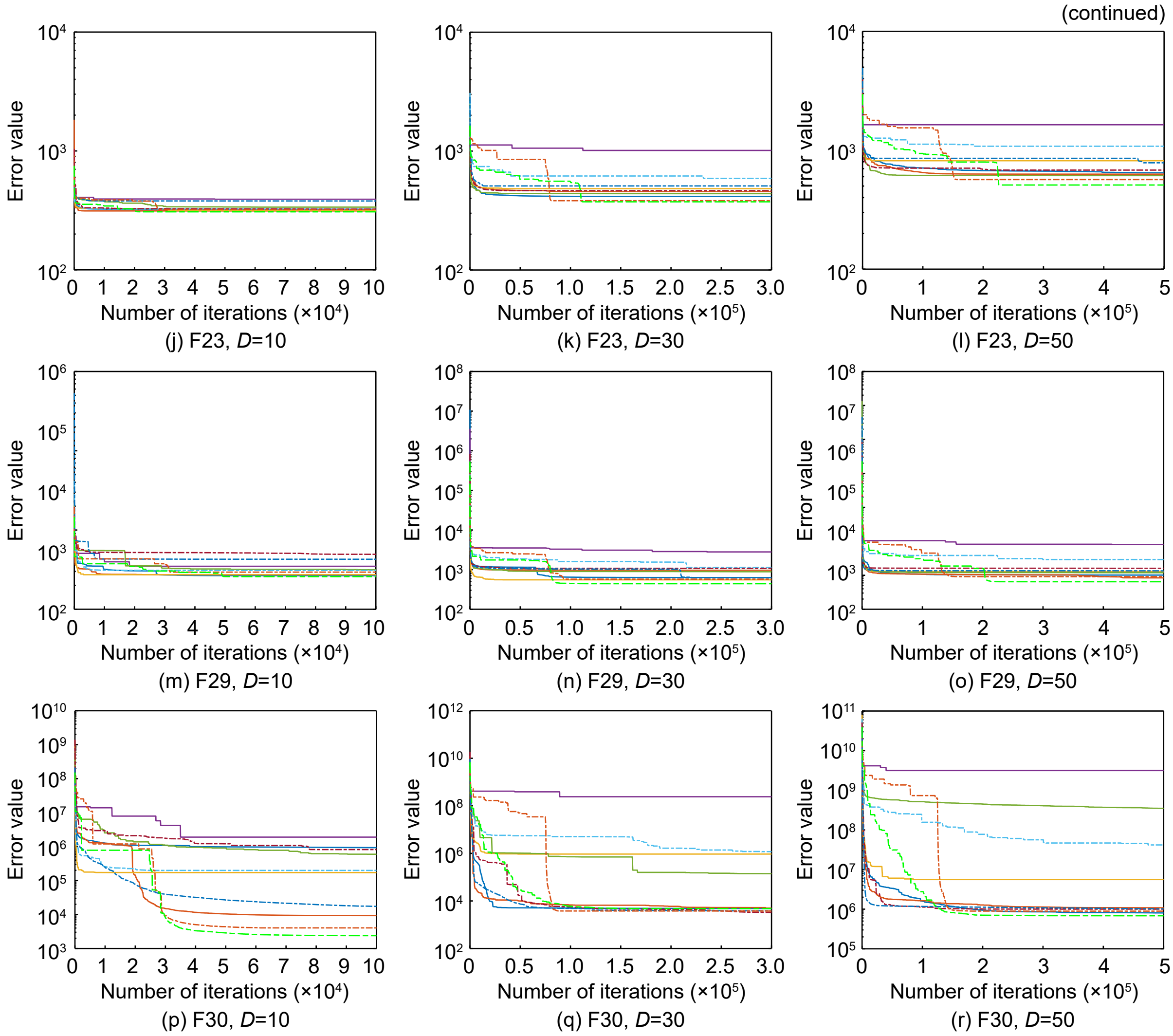


Fig. 3 Optimal error value iteration curve.

Tables 3−8 highlight the exceptional performance of the DMDS-HS algorithm on the three unimodal functions, indicating its strong solution accuracy. Moreover, these findings also reveal that the DMDS-HS algorithm maintains a stable and outstanding performance as the dimensionality increases, showcasing its robustness in comparison to other algorithms. This provides confirmation of the remarkable results achieved by the DMDS-HS algorithm in solving complex optimization problems in high-dimensional spaces. Furthermore, the iteration plot in Fig. 3 shows that during the early stages of computation, the global random search of the DMDS-HS algorithm effectively discovers potential global optima. As the iteration progresses, the algorithm gradually shifts towards local search, resulting in accelerated convergence, enhanced solution quality, and improved search efficiency. In summary, the analysis of the experimental results highlights the strengths of the DMDS-HS algorithm. Its adaptive variation design of HMCR allows for a balance between global and local search strategies, leading to superior convergence rates and solution accuracy. The incorporation of dual-memory structure and dynamic trust region further enhances its ability to escape local optima. Additionally, the algorithm showcases robustness in high-dimensional scenarios. These findings provide valuable insights into the performance and capabilities of the DMDS-HS algorithm.

4.2 Analysis of computational complexity for DMDS-HS

The computational efficiency of an algorithm can be reflected by its computational intricacy. In this section, we will use the O notation to represent the computational intricacy of the EDMDS-HS for initialization, the establishment of the dual-memory, and the improvisation update process. The initialization computational intricacy of UHM and LHM is $O(\text{HMS}\times D)$, and equivalently $O(\text{HMS}\times D)$, where HMS is the size of UHM and LHM, and D represents the dimensionality of the optimization problem in terms of the number of decision variables. The computational intricacy for sorting all harmonies is $O(\text{HMS})$, and the improvisation process has a computational intricacy of $O(D)$ since sorting the HM is performed in each iteration, with a computational intricacy of $O(\text{HMS})$. Additionally, in the worst case, the computational intricacy of the update process is $O(\text{HMS})$. It should be noted that the computational complexities of parameter initialization, adaptive parameter calculation, and the update of UHM and LHM repositories are all $O(1)$. Therefore, if the maximum number of iterations is $T_{\max}$, the computational intricacy of the DMDS-HS algorithm will be less than $O(\text{HMS}\times D)+O((\text{HMS}+D+1)\times T_{\max})$, which can be considered as $O(\text{HMS}\times D) + O((\text{HMS}+D)\times T_{\max})$, similar to the computational intricacy of the HS algorithm.

4.3 Comparison of DMDS-HS in data clustering applications

The DMDS-HS algorithm has demonstrated exceptional performance in the CEC2017 benchmark functions. To validate its efficacy in overcoming real-world problems, we applied it to a data clustering problem and compared it against seven classical clustering algorithms and heuristic algorithms. The objective function and encoding scheme of the problem were adopted from Talaei et al.[56] In order to conduct a comprehensive evaluation, we selected 10 well-known clustering datasets and performed 51 experiments on each dataset with an equal number of evaluations (10 000 iterations). The results of the experiments are presented in Table 2. The findings indicate that the DMDS-HS algorithm outperforms K-means, K-means++, GA, PSO, DE, SCA, and HS algorithms in 7, 7, 10, 10, 8, 10, and 9 of the 10 datasets, respectively. These seven datasets encompass diverse data characteristics and distributions, and the DMDS-HS algorithm exhibits strong clustering performance on these complex datasets. This highlights the algorithm's robust adaptability and versatility in effectively handling various types of data.

5 Conclusion

We propose the DMDS-HS algorithm as an enhancement to the HS algorithm, aiming to improve its performance. DMDS-HS incorporates a dual memory structure and dynamic trust region to explore new harmonies and incorporates phased planning for global random search using Rule 3 of the harmony. By carefully designing the algorithm's parameters, the probabilities of using the improved rules are adjusted accordingly. After analyzing the experimental results of a large number of tests performed on the internationally standardized CEC2017 benchmark function set, we find that the DMDS-HS algorithm outperforms the other nine HS algorithms and four state-of-the-art heuristic algorithms in all dimensions. Furthermore, the algorithm has been applied to data clustering tasks, where it has demonstrated its effectiveness and reliability in solving complex clustering problems. In conclusion, the DMDS-HS algorithm showcases excellent performance in terms of diversity, escaping local optima, and solution accuracy by incorporating the dual memory structure and dynamic trust region. It maintains stability and outstanding performance when tackling optimization problems of different dimensions and function types. These results confirm the efficacy and practicality of the DMDS-HS algorithm, providing a viable approach for dealing with the real-world complex optimization problems.

Acknowledgment

This work was supported by the Fund of Innovative Training Program for College Students of Guangzhou University (No. s202211078116), Guangzhou City School Joint Fund Project (No. SL2022A03J01009), National Natural Science Foundation of China (No. 61806058), Natural Science Foundation of Guangdong Province (No. 2018A030310063), and Guangzhou Science and Technology Plan Project (No. 201804010 299).

References

[1] V. Krishnan and S. Katkoori, A genetic algorithm for the design space exploration of datapaths during high-level synthesis, *IEEE Trans. Evol. Comput.*, vol. 10, no. 3, pp. 213–229, 2006.

[2] A. M. Brintrup, J. Ramsden, H. Takagi, and A. Tiwari, Ergonomic chair design by fusing qualitative and quantitative criteria using interactive genetic algorithms, *IEEE Trans. Evol. Comput.*, vol. 12, no. 3, pp. 343–354, 2008.

[3] Á. Rubio-Largo, L. Vanneschi, M. Castelli, and M. A. Vega-Rodríguez, Multiobjective metaheuristic to design RNA sequences, *IEEE Trans. Evol. Comput.*, vol. 23, no.

1, pp. 156–169, 2019.

[4] A. Tiwari, V. Oduguwa, and R. Roy, Rolling system design using evolutionary sequential process optimization, *IEEE Trans. Evol. Comput.*, vol. 12, no. 2, pp. 196–202, 2008.

[5] X. Yan, H. Zuo, C. Hu, W. Gong, and V. S. Sheng, Load optimization scheduling of chip mounter based on hybrid adaptive optimization algorithm, *Complex System Modeling and Simulation*, vol. 3, no. 1, pp. 1–11, 2023.

[6] Z. Shu, A. Song, G. Wu, and W. Pedrycz, Variable reduction strategy integrated variable neighborhood search and NSGA-II hybrid algorithm for emergency material scheduling, *Complex System Modeling and Simulation*, vol. 3, no. 2, pp. 83–101, 2023.

[7] H. Bai, T. Fan, Y. Niu, and Z. Cui, Multi-UAV cooperative trajectory planning based on many-objective evolutionary algorithm, *Complex System Modeling and Simulation*, vol. 2, no. 2, pp. 130–141, 2022.

[8] X. Shen, J. Lu, X. You, L. Song, and Z. Ge, A region enhanced discrete multi-objective fireworks algorithm for low-carbon vehicle routing problem, *Complex System Modeling and Simulation*, vol. 2, no. 2, pp. 142–155, 2022.

[9] Y. Guo, Y. Huang, S. Ge, Y. Zhang, E. Jiang, B. Cheng, and S. Yang, Low-carbon routing based on improved artificial bee colony algorithm for electric trackless rubber-tyred vehicles, *Complex System Modeling and Simulation*, vol. 3, no. 3, pp. 169–190, 2023.

[10] X. Shen, H. Pan, Z. Ge, W. Chen, L. Song, and S. Wang, Energy-efficient multi-trip routing for municipal solid waste collection by contribution-based adaptive particle swarm optimization, *Complex System Modeling and Simulation*, vol. 3, no. 3, pp. 202–219, 2023.

[11] Z. W. Geem, J. H. Kim, and G. V. Loganathan, A new heuristic optimization algorithm: Harmony search, *Simulation*, vol. 76, no. 2, pp. 60–68, 2001.

[12] J. Gholami, F. Pourpanah, and X. Wang, Feature selection based on improved binary global harmony search for data classification, *Appl. Soft Comput.*, vol. 93, p. 106402, 2020.

[13] J. Yi, C. H. Chu, C. L. Kuo, X. Li, and L. Gao, Optimized tool path planning for five-axis flank milling of ruled surfaces using geometric decomposition strategy and multi-population harmony search algorithm, *Appl. Soft Comput.*, vol. 73, pp. 547–561, 2018.

[14] E. Khorram and M. Jaberipour, Harmony search algorithm for solving combined heat and power economic dispatch problems, *Energy Convers. Manag.*, vol. 52, no. 2, pp. 1550–1554, 2011.

[15] Z. Li, D. Zou, and Z. Kong, A harmony search variant and a useful constraint handling method for the dynamic economic emission dispatch problems considering transmission loss, *Eng. Appl. Artif. Intell.*, vol. 84, pp. 18–40, 2019.

[16] L. D. S. Coelho and V. C. Mariani, An improved harmony search algorithm for power economic load dispatch, *Energy Convers. Manag.*, vol. 50, no. 10, pp. 2522–2526, 2009.

[17] I. A. Doush, M. A. Al-Betar, M. A. Awadallah, E. Santos, A. I. Hammouri, M. Mafarjeh, and Z. AlMeraj, Flow shop scheduling with blocking using modified harmony search algorithm with neighboring heuristics methods, *Appl. Soft Comput.*, vol. 85, p. 105861, 2019.

[18] K. Z. Gao, P. N. Suganthan, Q. K. Pan, T. J. Chua, T. X. Cai, and C. S. Chong, Pareto-based grouping discrete harmony search algorithm for multi-objective flexible job shop scheduling, *Inf. Sci.*, vol. 289, pp. 76–90, 2014.

[19] K. Z. Gao, P. N. Suganthan, Q. K. Pan, T. J. Chua, T. X. Cai, and C. S. Chong, Discrete harmony search algorithm for flexible job shop scheduling problem with multiple objectives, *J. Intell. Manuf.*, vol. 27, no. 2, pp. 363–374, 2016.

[20] S. Kulluk, L. Ozbakir, and A. Baykasoglu, Training neural networks with harmony search algorithms for classification problems, *Eng. Appl. Artif. Intell.*, vol. 25, no. 1, pp. 11–19, 2012.

[21] S. H. Kim, Z. W. Geem, and G. T. Han, Hyperparameter optimization method based on harmony search algorithm to improve performance of 1D CNN human respiration pattern recognition system, *Sensors*, vol. 20, no. 13, p. 3697, 2020.

[22] Z. Jia, Prediction of college students' psychological crisis with a neural network optimized by harmony search algorithm, *Int. J. Emerg. Technol. Learn.*, vol. 17, no. 2, pp. 59–75, 2022.

[23] M. Özçalıcı, A. T. Dosdoğru, A. B. İpek, and M. Göçken, Comparison of harmony search derivatives for artificial neural network parameter optimisation: Stock price forecasting, *Int. J. Data Min. Model. Manag.*, vol. 14, no. 4, pp. 335–357, 2022.

[24] G. Shen, Research on marine water quality evaluation model based on improved harmony search algorithm by Gaussian disturbance to optimize takagi-sugeno fuzzy neural network, *J. Coast. Res.*, vol. 111, no. sp1, pp. 283–287, 2020.

[25] O. Ceylan and G. Taşkın, SVM parameter selection based on harmony search with an application to hyperspectral image classification, in *Proc. 2016 24th Signal Processing and Communication Application Conference* (*SIU*), Zonguldak, Türkiye, 2016, pp. 657–660.

[26] X. Li, X. Li, and G. Yang, A novelty harmony search algorithm of image segmentation for multilevel thresholding using learning experience and search space constraints, *Multimed. Tools Appl.*, vol. 82, no. 1, pp. 703–723, 2023.

[27] Shivali, L. Maurya, E. Sharma, P. Mahapatra, and A. Doegar, A hybrid of fireworks and harmony search algorithm for multilevel image thresholding, in *Advanced Computing and Communication Technologies*, R. K. Choudhary, J. K. Mandal, and D. Bhattacharyya, eds. Singapore: Springer, 2018, pp. 11–21.

[28] R. Srikanth and K. Bikshalu, Multilevel thresholding image segmentation based on energy curve with harmony Search Algorithm, *Ain Shams Eng. J.*, vol. 12, no. 1, pp. 1–20, 2021.

[29] C. Peraza, F. Valdez, and O. Castillo, Interval type-2 fuzzy logic for dynamic parameter adaptation in the harmony

search algorithm, in *Proc. 2016 IEEE 8th Int. Conf. Intelligent Systems* (*IS*), Sofia, Bulgaria, 2016, pp. 106–112.

[30] M. Shaqfa and Z. Orbán, Modified parameter-setting-free harmony search (PSFHS) algorithm for optimizing the design of reinforced concrete beams, *Struct. Multidiscip. Optim.*, vol. 60, no. 3, pp. 999–1019, 2019.

[31] Y. W. Jeong, S. M. Park, Z. W. Geem, and K. B. Sim, Advanced parameter-setting-free harmony search algorithm, *Appl. Sci*., vol. 10, no. 7, p. 2586, 2020.

[32] F. Valdez, O. Castillo, and C. Peraza, Fuzzy logic in dynamic parameter adaptation of harmony search optimization for benchmark functions and fuzzy controllers, *Int. J. Fuzzy Syst.*, vol. 22, no. 4, pp. 1198–1211, 2020.

[33] A. Ocak, S. M. Nigdeli, G. Bekdaş, S. Kim, and Z. W. Geem, Adaptive harmony search for tuned liquid damper optimization under seismic excitation, *Appl. Sci.*, vol. 12, no. 5, p. 2645, 2022.

[34] U. Boryczka and K. Szwarc, The harmony search algorithm with additional improvement of harmony memory for asymmetric traveling salesman problem, *Expert Syst. Appl.*, vol. 122, pp. 43–53, 2019.

[35] J. Yi, X. Li, C. H. Chu, and L. Gao, Parallel chaotic local search enhanced harmony search algorithm for engineering design optimization, *J. Intell. Manuf.*, vol. 30, no. 1, pp. 405–428, 2019.

[36] M. Wang, T. Zhang, P. Wang, and X. Chen, An improved harmony search algorithm for solving day-ahead dispatch optimization problems of integrated energy systems considering time-series constraints, *Energy Build.*, vol. 229, p. 110477, 2020.

[37] F. Amini and P. Ghaderi, Hybridization of Harmony Search and Ant Colony Optimization for optimal locating of structural dampers, *Appl. Soft Comput.*, vol. 13, no. 5, pp. 2272–2280, 2013.

[38] M. Gheisarnejad, An effective hybrid harmony search and cuckoo optimization algorithm based fuzzy PID controller for load frequency control, *Appl. Soft Comput.*, vol. 65, pp. 121–138, 2018.

[39] A. E. Kayabekir, Y. C. Toklu, G. Bekdaş, S. M. Nigdeli, M. Yücel, and Z. W. Geem, A novel hybrid harmony search approach for the analysis of plane stress systems via total potential optimization, *Appl. Sci.*, vol. 10, no. 7, p. 2301, 2020.

[40] A. Radman, Combination of BESO and harmony search for topology optimization of microstructures for materials, *Appl. Math. Model.*, vol. 90, pp. 650–661, 2021.

[41] J. Gong, Z. Zhang, J. Liu, C. Guan, and S. Liu, Hybrid algorithm of harmony search for dynamic parallel row ordering problem, *J. Manuf. Syst.*, vol. 58, pp. 159–175, 2021.

[42] Q. K. Pan, P. N. Suganthan, M. F. Tasgetiren, and J. J. Liang, A self-adaptive global best harmony search algorithm for continuous optimization problems, *Appl. Math. Comput.*, vol. 216, no. 3, pp. 830–848, 2010.

[43] M. Mahdavi, M. Fesanghary, and E. Damangir, An improved harmony search algorithm for solving optimization problems, *Appl. Math. Comput.*, vol. 188, no. 2, pp. 1567–1579, 2007.

[44] M. G. H. Omran and M. Mahdavi, Global-best harmony search, *Appl. Math. Comput.*, vol. 198, no. 2, pp. 643–656, 2008.

[45] D. Zou, L. Gao, J. Wu, and S. Li, Novel global harmony search algorithm for unconstrained problems, *Neurocomputing*, vol. 73, nos. 16–18, pp. 3308–3318, 2010.

[46] E. Valian, S. Tavakoli, and S. Mohanna, An intelligent global harmony search approach to continuous optimization problems, *Appl. Math. Comput.*, vol. 232, pp. 670–684, 2014.

[47] H. B. Ouyang, L. Q. Gao, S. Li, X. Y. Kong, Q. Wang, and D. X. Zou, Improved harmony search algorithm, *Appl. Soft Comput.*, vol. 53, pp. 133–167, 2017.

[48] J. Gholami, K. K. A. Ghany, and H. M. Zawbaa, A novel global harmony search algorithm for solving numerical optimizations, *Soft Comput. A Fusion Found. Methodol. Appl.*, vol. 25, no. 4, pp. 2837–2849, 2021.

[49] Q. Zhu, X. Tang, Y. Li, and M. O. Yeboah, An improved differential-based harmony search algorithm with linear dynamic domain, *Knowledge-Based Systems*, vol. 187, p. 104809, 2020.

[50] H. Qing and S. H. Luo, Chimp optimization algorithm based on hybrid improvement strategy and its mechanical application, (in Chinese), *Control and Decision*, vol. 38, no. 2, pp. 354–364, 2023.

[51] Y. F. Wang, R. H. Liao, E. H. Liang, and J. W. Sun, Improved whale optimization algorithm based on siege mechanism, (in Chinese), *Control and Decision*, vol. 38, no. 10, pp. 2773–2782, 2023.

[52] Q. Y. Li and Y. X. Shen, A hybrid gray wolf optimization algorithm based on the teaching-learning optimization, (in Chinese), *Control and Decision*, vol. 37, no. 12, pp. 3190–3196, 2022.

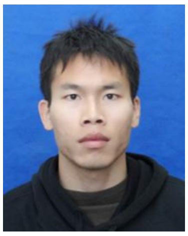
[53] S. Mirjalili, S. M. Mirjalili, and A. Lewis, Grey wolf optimizer, *Adv. Eng. Softw.*, vol. 69, pp. 46–61, 2014.

[54] S. Kotz and N. L. Johnson, *Breakthroughs in Statistics: Methodology and Distribution*. New York, NY, USA: Springer, 1992.

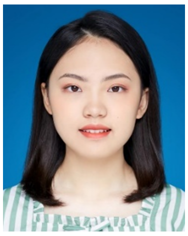
[55] H. B. Mann and D. R. Whitney, On a test of whether one of two random variables is stochastically larger than the other, *Ann. Math. Statist.*, vol. 18, no. 1, pp. 50–60, 1947.

[56] K. Talaei, A. Rahati, and L. Idoumghar, A novel harmony search algorithm and its application to data clustering, *Appl. Soft Comput.*, vol. 92, p. 106273, 2020.

Jinglin Wang is currently pursuing the bachelor degree in robotics engineering at Guangzhou University, China. His research interests include intelligent optimization algorithm and optimal control.



Haibin Ouyang received the MS and PhD degrees in control theory and control engineering from Northeastern University (NEU), Shenyang, China in 2012 and 2016, respectively. Currently, he is an associate professor at the School of Mechanical and Electric Engineering, Guangzhou University, Guangzhou, China. He has published over 50 papers in several journals, including *Information Sciences*, *Applied Soft Computing*, *Mathematics and Computers in Simulation*, and *Applied Mathematics and Computation*. His current research interests are intelligent optimization algorithm, robotics path planning, artificial intelligence, and optimal control. He is the editorial board member of *Applied Soft Computing*.



Zhiyu Zhou is currently pursuing the bachelor degree in robotics engineering at Guangzhou University, China. Her research interests include intelligent optimization algorithm and transfer learning.

Steven Li received the PhD degree from Delft University of Technology, the Netherlands in 1992, the MBA degree from The University of Melbourne, Australia in 2000, and the BS degree from Tsinghua University, China in 1987. He is a professor of finance at RMIT University, Melbourne, Australia. He previously taught at University of South Australia, Queensland University of Technology, Edith Cowan University, and Tsinghua University. His current research interests are mainly in quantitative finance, financial management, and intelligent algorithm and its application. He has published extensively over 80 publications in international journals including *Journal of International Financial Markets*, *European Journal of Finance*, *International Review of Economics and Finance*, *Applied Energy*, *Applied Intelligence*, etc.