applied sciences

MDPI

Article

A New Differential Mutation Based Adaptive Harmony Search Algorithm for Global Optimization

Xinchao Zhao [1,2,*], Rui Li [1], Junling Hao [3], Zhaohua Liu [1] and Jianmei Yuan [2,4,*]

1 School of Science, Beijing University of Posts and Telecommunications, Beijing 100876, China; zmdsn@126.com (R.L.)
2 Hunan Key Laboratory for Computation and Simulation in Science and Engineering, Xiangtan University, Xiangtan 411105, China
3 School of Statistics, University of International Business and Economics, Beijing 10029, China
4 School of Mathematics and Computational Science, Xiangtan University, Xiangtan 411105, China
* Correspondence: zhaoxc@bupt.edu.cn (X.Z.); yuanjm@xtu.edu.cn (J.Y.)

Received: 22 February 2020; Accepted: 17 April 2020; Published: 23 April 2020

Abstract: The canonical harmony search (HS) algorithm generates a new solution by using random adjustment. However, the beneficial effects of harmony memory are not well considered. In order to make full use of harmony memory to generate new solutions, this paper proposes a new adaptive harmony search algorithm (aHSDE) with a differential mutation, periodic learning and linear population size reduction strategy for global optimization. Differential mutation is used for pitch adjustment, which provides a promising direction guidance to adjust the bandwidth. To balance the diversity and convergence of harmony memory, a linear reducing strategy of harmony memory is proposed with iterations. Meanwhile, periodic learning is used to adaptively modify the pitch adjusting rate and the scaling factor to improve the adaptability of the algorithm. The effects and the cooperation of the proposed strategies and the key parameters are analyzed in detail. Experimental comparison among well-known HS variants and several state-of-the-art evolutionary algorithms on CEC 2014 benchmark indicates that the aHSDE has a very competitive performance.

Keywords: harmony search; differential mutation; population size reduction; periodic learning

1. Introduction

The Harmony Search (HS) algorithm is one of the Evolutionary Algorithms (EA), taking inspiration from the music improvisation process, which was proposed by Geem et al. [1] in 2001. It is an emerging population-based metaheuristic optimization algorithm which simulates the improvisation behavior of musicians by repeatedly adjusting the instruments, eventually generating a harmony state. In HS, the harmony of musical instrument tones is regarded as a solution vector of the optimization problem. The evaluation of musical harmonies corresponds to the objective function value.

There are four main control parameters in a canonical HS algorithm [1], including the harmony memory size (*HMS*), harmony memory considering rate (*HMCR*), pitch adjusting rate (*PAR*) and the bandwidth (*bw*). However, it is well known that the optimal setting of these parameters [2] depends on the problem. Therefore, when HS is applied to real-world problems, it is necessary to adjust the control parameters to obtain the desired results. Overall, it has attracted more and more attention and a variety of HS variants have been proposed.

In order to improve its efficiency or to overcome some shortcomings, the original HS operators have been adapted and/or new operators have been introduced. Mahdavi et al. [3] proposed an improved harmony search algorithm (IHS) in which PAR is designed to increase linearly, while *bw* decreases exponentially with the increase of the number of iterations. Pan et al. [4] proposed a

self-adaptive global-best harmony search (SGHS). It employs a new improvisation scheme and uses a parameter adjustment strategy to generate a new solution with a learning period. Combining the harmony search algorithm with particle swarm optimization (PSO) [5], Valian et al. [6] presented an intelligent global harmony search algorithm (IGHS) which has excellent performance compared with its competitors. To enhance the search efficiency and effectiveness, a self-adaptive global-best harmony search algorithm [7] is developed. The proposed algorithm takes full advantage of the valuable information hidden in the harmony memory to devise a high-performance search strategy. It also integrates a self-adaptive mechanism to develop a parameter-setting-free technique [8]. Ouyang et al. [9] proposed an improved harmony search algorithm with three key features: adaptive global pitch adjustment, opposition-based learning and competition selection mechanism. Inspired from the simulated annealing of accepted inferior solutions, a hybrid harmony search algorithm (HSA) [10] is proposed. It accepts the inferior harmonies with a probability determined by a temperature parameter. Zhu et al. [11] proposed an improved differential-based harmony search algorithm with a linear dynamic domain which utilized two main innovative strategies. Focusing on the historical development of algorithm structures, Zhang and Geem [12] reviewed various modified and hybrid HS methods, which included the adaption of the original operators, parameter adaption, hybrid methods, handling multi-objective optimization and constraint handling.

One question is naturally proposed—why does HS work on various problems from science and engineering [13]? The unique stochastic derivative [14] gives information on the probabilistic inclination to select certain discrete points based on multiple vectors stored in *HM* for a discrete problem. Although HS is easy to implement and has a simple structure [13], it has shown its superiority with more complex optimization algorithms, and has been applied to many practical problems [12,15–17]. HS has been successfully used in a wide range of applications [18–25], which has attracted a lot of research attention undertaken to further improve its performance. Combining HS and local search, a novel sensor localization approach is proposed by Manjarres et al. [26]. Minimizing energy consumption and maximizing the network lifetime of a wireless sensor network (WSN) problem using the HS algorithm was closely studied [27–30]. Degertekin [31] optimized the frame size of truss structures by harmony search algorithm. Compared with the genetic algorithm for max-cut problem [32], the harmony search algorithm has advantages of generating new vectors after considering all of the existing vectors and parameters. Boryczka and Szwarc [33] proposed a harmony search algorithm with the additional improvement of harmony memory for asymmetric traveling salesman problems, which eliminates the imperfectness revealed in the previous research. Seyedhosseini et al. [34] researched the portfolio optimization problem using a mean-semi variance approach based on harmony search and an artificial bee colony. HSA [35] was used in reservoir engineering-assisted history, matching questions of different complex degrees, which are two material balance history matches of different scales and one reservoir history matching.

However, HS and its variants usually have the following drawbacks, which are also our research motivation.

(1) For the pitch adjustment operator of HS, a larger bandwidth is easier to jump out of the local optimum, while a smaller bandwidth biases to find a promising solution for the fining search. Therefore, a fixed step size is not an ideal choice.
(2) It is difficult to find the optimal solution with a constant execution probability and an adaptive adjusting method is required.
(3) Parameter *HMS* has an important influence on the performance of algorithms. An adaptive sizing *HMS* is possible to enhance the performance of the algorithm.

Therefore, an adaptive harmony search algorithm is proposed with differential evolution mutation, periodic learning and linear population size reduction (aHSDE). The main contributions of this paper are as follows.

(1) The pitch adjustment strategy is implemented with differential mutation. Adjust the pitch adjusting rate *PAR* and the scaling factor *F* with periodic learning strategy. Linear population size reduction strategy is adopted for *HMS* changing scheme.
(2) The cooperation and effects of several strategies are analyzed step by step.

The organization of this paper is as follows. Section 2 reviews the canonical HS and several improved variants. In Section 3, the composite strategies and algorithm aHSDE are proposed. In Section 4, the effects and the cooperation of the proposed strategies and parameters analysis are presented. The comprehensive performance comparison with other HS variants and other state-of-the-art EAs are presented in Section 5. Finally, Section 6 concludes the paper.

2. Harmony Search and Several Variants

2.1. Harmony Search Algorithm

The harmony search algorithm is a new population-based metaheuristic optimization algorithm [1], which is inspired by the improvisation process of music. The improvisation process is modeled as an iterative optimization method and the musicians' musical instruments improvise to produce a better harmony [36]. The basic steps are described in detail in Algorithm 1.

Algorithm 1: General Framework of the Harmony Search (HS).

1: *//Initialize the problem and algorithm parameters//*
$f(x)$*:* objective function
HMS: harmony memory size
HMCR: harmony memory considering rate
PAR: pitch adjusting rate
bw: bandwidth
DIM: dimension of decision variable
MAX_NFE: maximum number of function evaluations
L_i, U_i: the lower and upper bounds of the *i*-th component for the decision vector

2: *//Initialize the harmony memory* $(x^1, x^2, \cdots, x^{HMS})$*//*
$x_i^j = L_i + rand(0,1) \cdot (U_i - L_i)$

3: *//Improvise a new harmony* $x^{new} = (x_1^{new}, x_2^{new}, \cdots, x_{DIM}^{new})$*//*
$for\ each\ i \in [1, DIM]\ do$
$if\ (rand(0,1) < HMCR)\ then$
$x_i^{new} = x_i^a, where\ a \in \{1,2,\ldots,HMS\}$
$if(rand(0,1) < PAR)\ then$
$x_i^{new} = x_i^{new} + rand(-1,1) \times bw$
$endif$
$else$
$x_i^{new} = L_i + rand \cdot (U_i - L_i)$
$endif$
$endfor$

4: *//Update the harmony memory//*
$if\ f(x^{new}) < f(x^{worst}) = \max_{j=1,2,\ldots,HMS} f(x^j),\quad then\quad x^{worst} = x^{new}.$

5: *//Check the stopping criterion//*
If the termination condition is met, stop and output the best individual. Otherwise, the process will repeat from Step 3.

2.2. The Improved Harmony Search Algorithm (IHS)

In the canonical harmony search algorithm, the probability of *PAR* and *bw* are constant. Mahdavi et al. [3] proposed an improved harmony search algorithm, called IHS, which mainly introduced the dynamic change of *PAR* and *bw* using the following equations.

$$PAR(NFE) = PAR_{min} + \frac{PAR_{max} - PAR_{min}}{MAX_{NFE}} \times NFE \quad (1)$$

$$bw(NFE) = bw_{max} exp\left(\frac{ln\frac{bw_{min}}{bw_{max}}}{MAX_{NFE}} \times NFE\right) \quad (2)$$

where PAR_{max} is the maximum adjusting rate; PAR_{min} is the minimum adjusting rate; bw_{max} is the maximum bandwidth; bw_{min} is the minimum bandwidth. MAX_{NFE} is the maximum number of function evaluation and *NFE* is the current number of function evaluation.

2.3. A Self-Adaptive Global-Best Harmony Search (SGHS)

The SGHS [4] employs a new improvisation scheme and an adaptive parameter tuning method. To modify pitch adjustment rule, x_i^{new} is assigned with the corresponding decision variable x_i^{best} from the best harmony vector. In addition, the concept of a learning period is introduced. Parameters HMCR and PAR are dynamically adapted to a suitable range by recording their historical values corresponding to the generated successful harmonies entering the nest harmony memory. Furthermore, *bw* is dynamically updated using the following equation.

$$bw(NFE) = \begin{cases} bw_{max} - \frac{bw_{max} - bw_{min}}{MAX_{NFE}} \times 2NFE, & if\ NFE < \frac{MAX_{NFE}}{2} \\ bw_{min}, & if\ \ NFE > \frac{MAX_{NFE}}{2} \end{cases} \quad (3)$$

where $bw_{\max}$ and $bw_{\min}$ are the maximum and minimum values of the bandwidth (*bw*), respectively.

2.4. An Intelligent Global Harmony Search Algorithm (IGHS)

Valian et al. [6] modified the improvisation step by imitating one dimension of the best harmony in the harmony memory to generate the new harmony and proposed algorithm IGHS. The main steps are shown in Algorithm 2.

Algorithm 2: Main framework of the intelligent global harmony search algorithm (IGHS).

```
for each i ∈ [1, DIM] do
  if (rand(0,1) < HMCR) then
    if(rand(0,1) < PAR) then
      x_i^new = x_k^best ,      where k ∈ {1,2, … , DIM}
        else
      x_i^R = 2 × x_i^best − x_i^worst
      if  x_i^R < L_i
       x_i^R = L_i
      elseif  x_i^R > U_i
       x_i^R = U_i
            endif
               x_i^new = x_i^worst + rand · (x^R − x^worst)
          endif
  else
          x_i^new = L_i + rand · (U_i − L_i)
    endif
endfor
```

3. Adaptive Harmony Search with Differential Evolution

When musicians compose music, they take full advantage of their own knowledge and experience in the improvement direction. With the continuous optimization of music composition, musicians will also reduce the available experience and accelerate the composition. Inspired by the conception, it is desired to make full use of the information in the harmony memory and dynamically adjust the harmony memory size. Thus, the differential evolution mutation is adopted into the modified algorithm to provide an effective guidance for the generation of the new solution. The linear harmony memory size reduction strategy is also introduced into the algorithm to accelerate the convergence. Meanwhile, in order to strengthen the general suitability for various problems and reduce the dependence on the parameters, the parameter's self-adaption with the concept of a learning period is applied to the modified algorithm.

This paper presents an adaptive harmony search algorithm (aHSDE) with differential evolution mutation, periodic learning and linear population size reduction strategy. Therefore, it is desirable to balance the ability of global exploration and local exploitation for the harmony search algorithm.

3.1. Differential Evolution

As a stochastic population optimization algorithm, differential evolution (DE) is similar to other evolutionary algorithms [37]. The basic idea of DE is summarized as follows: a set of initial individuals are generated randomly in the search space, and each individual represents a solution. After this, a new individual is generated by the following three operations in sequence: mutation, crossover and selection. The core idea of DE is that it adds the differential vectors among several individual pairs to a base vector. It controls the magnitude and direction of exploration for the promising neighborhood [38].

This paper uses DE/best/2 of DE mutation as follows:

$$x_i^{new} = x_i^{best} + F[(x_i^{r_1} - x_i^{r_2}) + (x_i^{r_3} - x_i^{r_4})] \tag{4}$$

where r_1, r_2, r_3 *and* r_4 are mutually different individual indexes which are chosen randomly. The parameter F is a scale factor controlling the mutation step size. Scaled differential vectors with respect to the possible individual pairs adapt the property of the current neighborhood landscape. It thus can provide promising mutation directions with adjustable step size and a balance between local and global search [39].

3.2. Linear Population Size Reduction

In the former search stage of HS, the algorithm tends to explore the search space with the assistance of well-population diversity. Subsequently, it can construct some fine-tuning directions in the iterative process. In the latter stage of the algorithm, the population usually focuses on the neighbor search. Therefore, exploitation better attracts most of the computing resource. Inspired by the improved Success-History based parameter Adaptation for Differential Evolution (SHADE) [40], a monotonically reducing population size strategy is utilized with respect to the function evaluation number. It is shown as follows

$$HMS = round\left(HMS_{\max} - \frac{HMS_{\max} - HMS_{\min}}{MAX_{NFE}} \times NFE\right) \tag{5}$$

where $HMS_{\max}$ and $HMS_{\min}$ are the maximum and minimum values of the harmony memory size (HMS), respectively. MAX_{NFE} is the maximum number of function evaluation and NFE is the current number of function evaluation.

3.3. Differential Mutation in the Pitch Adjustment Operator

The canonical harmony search algorithm operates the pitch adjustment with the constant distance bandwidth, which cannot adapt to the searching landscapes at different searching stages for different problems. It is certain that a proper bandwidth is important for the harmony search algorithm. In this

paper, we present a general framework for defining the pitch adjustment operator with the differential mutation (DE/best/2) [41], which can provide a more effective direction than the constant bandwidth to the searching landscape. It is indicated as Equation (6).

$$x_i^{new} = x_i^{best} + F[(x_i^{r_1} - x_i^{r_2}) + (x_i^{r_3} - x_i^{r_4})] + rand(-1,1) \times bw \quad (6)$$

where i is the component index from $\{1, 2, 3, \ldots, DIM\}$; r_1, r_2, r_3, r_4 are selected randomly from $\{1, 2, 3, \ldots, DIM\}$ and mutually exclusive; rand(−1,1) is a uniformly distributed random number between −1 and 1. DIM is the dimension of decision variables. x^{new} is the newly generated harmony vector and x^{best} is the current best harmony vector in the harmony memory.

If the new solution is out of the bounds $[L_i, U_i]$, it will be modified as follows,

$$x_i^{new} = \begin{cases} L_i, & if \quad x_i^{new} < L_i \\ U_i, & if \quad x_i^{new} > U_i \end{cases} \quad (7)$$

where U_i and L_i denote the upper and lower bounds of the i-th component for the decision vector.

3.4. Self-Adaptive PAR and F

Inspired by the concept of the learning period of SGHS [21] to adaptively tune parameters HMCR and PAR, this paper employs a new modified scheme for *PAR* and *F*.

In addition, the parameters *HMCR* and *PAR* are dynamically adapted to a suitable range by recording their historic values corresponding to the generated harmonies entering the harmony memory. During the evolution, the values of *HMCR* and *PAR* for the generated successful harmony are recorded to replace the worst members in the harmony memory.

First, both the means of *PAR* (*PARm*) and *F* (*Fm*) are initialized as 0.5. Second, parameters *PAR* and *F* are generated with a normal distribution. During the generations, the values of *PAR* and *F* are recorded when the generated harmony successfully replaces the worst member in the harmony memory. After Learning Period (*LP*), parameters *PARm* and *Fm* are recalculated with the weighted Lehmer mean formulas [42]. The weighted Lehmer mean $mean_{wl}(S)$ uses the following deterministic Equations (8)–(10) to compute. The amount of fitness difference Δf_k is used to influence the parameter adaptation.

$$mean_{wl}(S) = \frac{\sum_{k=1}^{|S|} w^k \cdot S^{k2}}{\sum_{k=1}^{|S|} w^k \cdot S^k} \quad (8)$$

$$w^k = \frac{\Delta f^k}{\sum_{l=1}^{|S|} \Delta f^l} \quad (9)$$

$$\Delta f^k = \left| f(x^{new}) - f\left(x^{worst}\right) \right| \quad (10)$$

where S includes either S_{PAR} or S_F and $mean_{wl}(S)$ is the new value of *PARm* or *Fm*. x^{new} is the newly generated solution in the current generation. x^{worst} is the worst solution in harmony memory. $f(*)$ denotes the fitness function.

Parameters *PARm* or *Fm* use the framework in Algorithm 3 to update their values, where generation counter $lp = 1$. The difference between the weighted Lehmer mean and the arithmetic mean is described as follows. Arithmetic mean means all the recorded successful parameters of *PARm* or *Fm* have the same weights. On the other hand, the weighted Lehmer mean shown in Equations (8)–(10) means that all the recorded successful parameters of *PARm* or *Fm* have the self-adaptive weights based on their fitness improvements. It is very possible that the weighted Lehmer mean outperforms the arithmetic mean statistically. However, we will not analyze their difference in this paper due to paper length restrictions, and instead cite the reference [42] directly. The detailed information of weighted Lehmer mean can be found in reference [42].

Algorithm 3: **Parameters updating of the means of *PAR* (*PARm*) and *F* (*Fm*).**

if lp > LearnPeriod
$PARm = mean_{wl}(S_{PAR})$
$Fm = mean_{wl}(S_F)$
lp=1;
else
lp=lp+1
endif

In general, the values of *PAR* and *F* are regenerated as the following equations.

$$\begin{cases} PAR = normrnd(PARm, 0.1) \\ F = normrnd(Fm, 0.1) \end{cases} \quad (11)$$

If *PAR* is larger than 1, it is truncated to be 1; if *PAR* is less than or equal to 0, it will be assigned 0.001. The same action is executed on *F*.

3.5. aHSDE Algorithm Framework

The aim of this paper is to provide some beneficial strategies to improve the performance of the aHSDE from the improvisational perspective. Algorithm 4 shows the procedure of the aHSDE.

Algorithm 4: **Framework of the new adaptive harmony search algorithm (aHSDE).**

//Initialize the problem and parameters//
$f(x)$*:* objective function
HMSmax: the maximum value of the harmony memory size
HMSmin: the minimum value of the harmony memory size
HMCR: harmony memory considering rate
1: *PARm:* the mean of pitch adjusting rate
Fm: the mean of the scaled factor
bw: bandwidth
LP: learning period
MAX_NFE: maximum number of function evaluation
L, U: the lower and upper bounds of the decision vector
//Initialize the harmony memory $(x^1, x^2, \cdots, x^{HMS})$*//*
2: $x_i^j = L_i + rand(0,1) \cdot (U_i - L_i), j = 1,2, \ldots, HMS; i = 1,2, \ldots, DIM$
//Improvise a new harmony $x^{new} = (x_1^{new}, x_2^{new}, \cdots, x_{DIM}^{new})$*//*

(1) Update HMS with Equation (5). If HMS decreases, the solutions are sorted in HM according to their fitness values and the worst one is deleted.
(2) To generate a new solution, the process is as follows:

for each $i \in [1, DIM]$ *do*
if $(rand < HMCR)$ *then*
3: $x_i^{new} = x_i^{best}$
$if(rand(0,1) < PAR)$ *then*
$x_i^{new} = x_i^{best} + F[(x_i^{r_1} - x_i^{r_2}) + (x_i^{r_3} - x_i^{r_4})] \pm rand \times bw$
endif
else
$x_i^{new} = L_i + rand \cdot (U_i - L_i)$
endif
endfor
//Update the harmony memory//
4: *if* $f(x^{new}) < f(x^{worst}) = \max_{j=1,2,\ldots,HMS} f(x^j)$, *then* $x^{worst} = x^{new}$.
Record the generation of PAR, F and the fitness difference Δf^k.
//Check the stopping criterion//
5: *If the termination condition is met, stop and output the best individual.*
Otherwise, the process will repeat from Step 3.

4. Experimental Comparison and Analysis

The proposed strategies and the parameter adaption schemes are explained and analyzed firstly by empirical research in this section. Subsequently, the proposed aHSDE is compared with the classical HS and several state-of-the-art HS variants, which include IHS [3], SGHS [4] and IGHS [6]. It is also compared with other state-of-the-art evolutionary algorithms (non harmony ones), which are Adaptive Particle Swarm Optimization (APSO) [43] and Evolution Strategy with Covariance Matrix Adaptation (CMA-ES) [44].

4.1. Parameters and Benchmark Functions

This section evaluates the performance of the aHSDE on the CEC2014 benchmark suite [45] compared with the original HS, IHS, SGHS and IGHS. The CEC2014 benchmark suite consists of 30 test functions, which include three unimodal functions, 13 multimodal functions, six hybrid functions and eight composite functions. In particular, these hybrid functions (f17–f22) are very similar to real world problems, such as transportation networks [46], circuit theory [47], image processing [48], capacitated arc routing problems [49] and flexible job–shop scheduling problems [50]. The search range for each function is $[-100, 100]^{\mathrm{DIM}}$, where DIM is the dimension of the problem. The experiments are conducted in 10, 50 and 100 dimensions and the maximum function evaluation number is DIM $\times$ 10000. The number of multiple runs per function is 30, and the average results of these runs are recorded. When the value difference between the found best solution and the optimal solution is lower than 1×10^{-8}, the error is considered as 0.

For the comparing HS variants, the parameter settings are the same as the respective literatures, which are also shown in Algorithm 5.

Algorithm 5:	**Parameters.**
HS [1]	$\mathrm{HMS} = 5, \mathrm{HMCR} = 0.9, \mathrm{PAR} = 0.3, \mathrm{bw} = 0.01$
HIS [3]	$\mathrm{HMS} = 5, \mathrm{HMCR} = 0.9, \mathrm{PAR}_{min} = 0.01, \mathrm{PAR}_{max} = 0.99,$ $\mathrm{bw}_{min} = 0.0001, \mathrm{bw}_{max} = \frac{x^U - x^L}{20}$
SGHS [4]	$\mathrm{HMS} = 5, \mathrm{HMCR}_m = 0.98, \mathrm{PAR}_m = 0.9, \mathrm{Lp} = 100,$ $\mathrm{bw}_{min} = 0.0005, \mathrm{bw}_{max} = \frac{x^U - x^L}{10}$
IGHS [6]	$\mathrm{HMS} = 5, \mathrm{HMCR} = 0.995, \mathrm{PAR} = 0.4$
aHSDE	$\mathrm{HMS}_{min} = 5, \mathrm{HMS}_{max} = 18 \times \mathrm{DIM}, HMCR = 0.99, LP = 100, BW = 0.01$

4.2. How HMS Changes

In order to analyze the impact of the harmony memory size *HMS* on the aHSDE, four functions, f1, f10, f21, f28, are chosen from four categories, respectively. In the following experiments of Sections 4.2–4.5, the dimension size of four functions is 30 and the statistical results are obtained from 30 independent runs. The minimum value of HMS is five and the maximum value HMS_{max} is a maximal integer which is no larger than *rate* $\times$ DIM, which is associated with the problem dimension size. Furthermore, *rate* is considered changing from 0.5 to 25 with an interval 0.5. Then the best results in each case are recorded and shown in Figure 1.

As can be seen from Figure 1, the fitness of the four functions decreases exponentially when the initial value of *HMS* gradually changes from 0.5 $\times$ *DIM* to 25 $\times$ *DIM*. This indicates that the initial value of HMS has a great impact on the performance of the aHSDE. When the initial value is small, the fitness of function decreases rapidly. The decreasing trend is no longer clear for the fitness as the initial value increases to 15 $\times$ *DIM*. Therefore, the initial value of *HMS* is set to 18 $\times$ *DIM* in this paper without special explanation in the following sections.

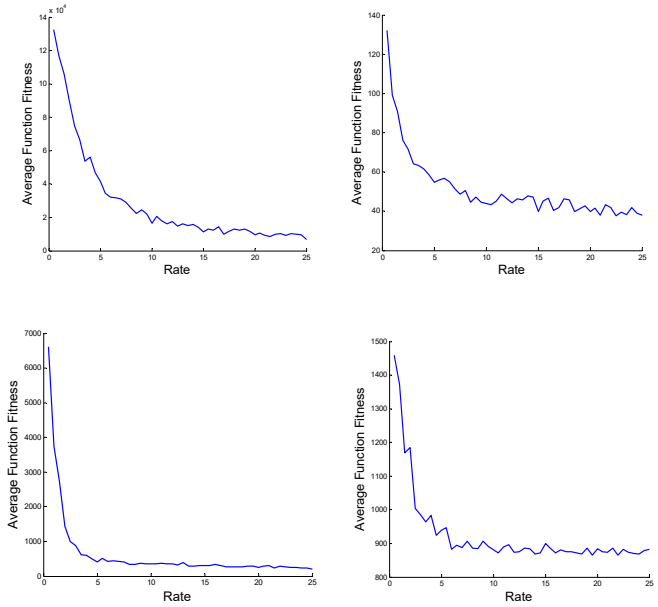

Figure 1. Impact of Harmony Memory Size (HMS) with 30 runs on the aHSDE (f1, f10, f21, f28).

4.3. Effect of Differential Evolution Based Mutation

In this paper, mutation strategy DE/best/2 is used to adjust bandwidth to explore the landscape of the corresponding sub-stages. Therefore, $\left((x_i^{r_1} - x_i^{r_2}) + (x_i^{r_3} - x_i^{r_4})\right)$, regarded as an Experience Operator (EO), is used to indicate the possible maximum searching neighborhood with the increase of generations. It can be adjusted adaptively with the change from population diverse to converging. The same analyzing functions, f1, f10, f21, f28, as detailed above, are chosen to illustrate the performance of the aHSDE algorithm with the increase of generations. Their changing trends of Eos are shown in Figure 2.

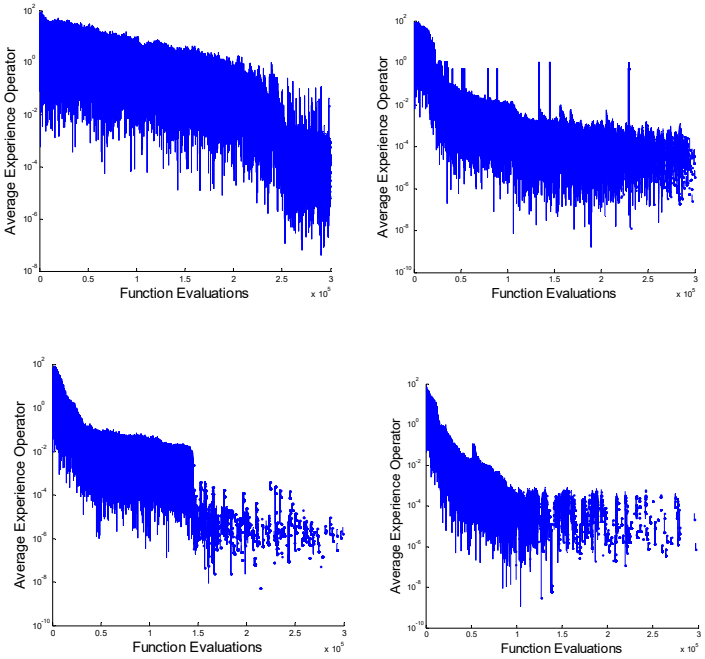

Figure 2. Change of Experience Operator on aHSDE (f1, f10, f21, f28).

It can be seen from Figure 2 that EO is gradually convergent with the increase of generations. In the early stage of iteration, the search region of the algorithm is relatively large and EO is relatively large accordingly, which strengthens the global exploration ability. However, with the gradual convergence of HMS, the fining search of the algorithm is gradually conducted to improve the local exploitation at the latter generations. It is possible to provide a promising escape mechanism from the landscape valley with the differential mutation strategy to adjust the pitch in the aHSDE. Therefore, the aHSDE precedes over the original HS and several HS variants, which exploits the valley using a small step size.

4.4. How PAR and F Change

In this paper, the concept of a learning period for *PAR* and *F* adjustment is adopted, which is computed with the weighted Lehmer mean. It aims to reduce the dependence on the parameters and enlarge the application scope of the algorithm. The results of *PAR* and *F* are recorded in 30 independent runs with the same four functions as the previous sections. Figures 3 and 4 illustrate the changing trends of the adaptive adjustment strategy for *PAR* and *F*.

Observed in Figure 3, *PAR* probably changes between 0.7 and 0.95, which is rather large in the early generations. However, it ranges around 0.1 in the latter generations. It is necessary that the aHSDE, as one of the population-based optimization algorithms, needs a wide neighborhood-based global exploration in a high probability in the early stage. After this, a probability of pitch adjustment becomes smaller and smaller in order to improve the fine-tuning search and convergence of the algorithm in the latter generations. Thus, this adaptive modification strategy for *PAR* is inherently consistent with the internal variation principle of the exploring step size for population based optimization algorithms. It is possible to keep a good balance between local exploitation and global exploration.

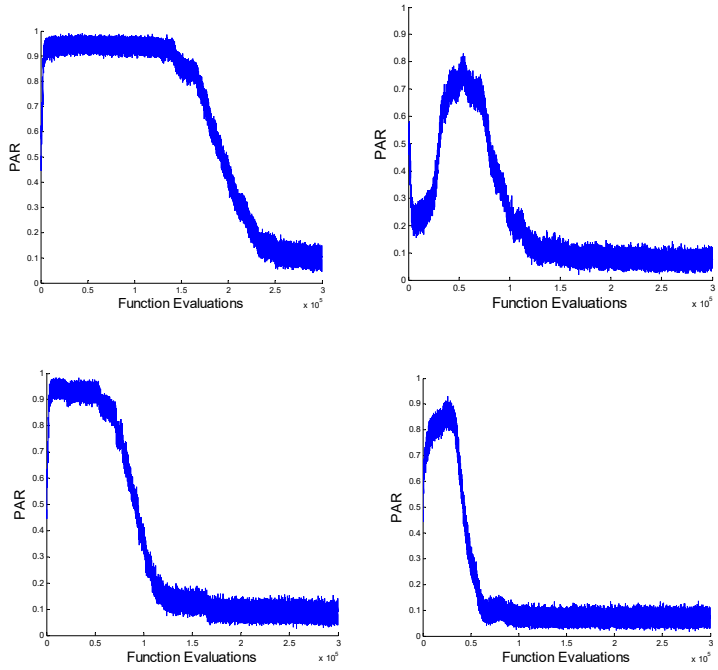

Figure 3. How pitch adjusting rate *(PAR)* changes f1, f10, f21, f28.

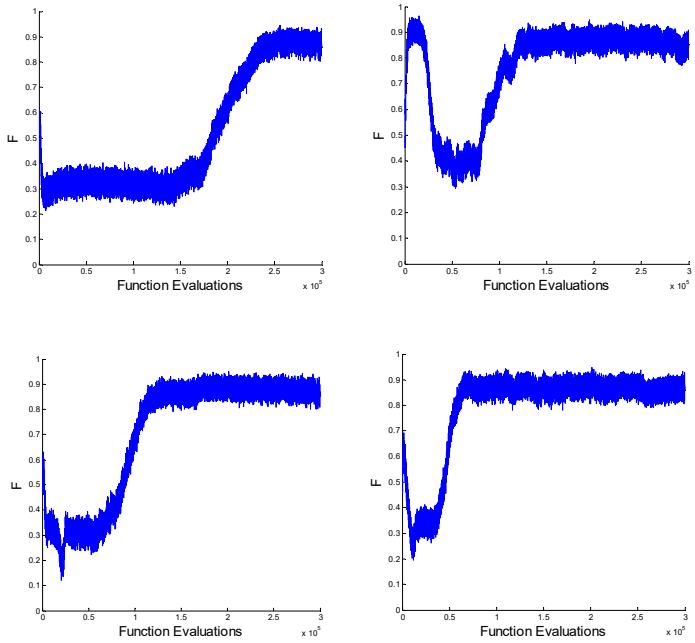

Figure 4. How the scaling factor *F* changes f1, f10, f21, f28.

It is observed that the overall changing trend of the scaling factor *F* for four functions is opposite to the parameter *PAR*. *F* is small at the beginning stage of iteration, at about 0.3. However, it becomes large at the latter iteration, probably around 0.9. It is worthy of note that the initial value of *F* is 0.5. Therefore, it can be roughly inferred that algorithm is not sensitive to the initial value of *F*. The possible reason for the four functions with similar trends is that the difference vector of the improving direction guided by DE operation is relatively large at the early generations. Therefore, a relatively small scaling factor *F* is suitable to the search demand for the algorithm. On the contrary, most of the solutions approximate to the optimal solution and the difference vector of improving direction is relatively small at the latter generations. Therefore, a large scaling factor *F* is required. In addition, although the overall changing trend of each function is similar, the adaptive adjustment behavior of *F* still depends on the solving problems. The curves, showing how *F* changes for four functions, exhibit different varying principles.

In this paper, the weighted Lehmer mean is used to adaptively tune *PAR* and scale the parameter *F*. It is a versatile and efficient automatic parameter tuner and is highly successful in tuning search and optimization algorithms [42].

4.5. Combined Adaptability Consideration for PAR and F

In order to consider the effects of parameters *PAR* and *F*, the same four functions are used to analyze the performance difference on the aHSDE with different parameter settings. The statistical results in 30 runs are shown in Table 1 and Tables S1–S3. The data in the Tables S1–S3 represent the statistical results of multiple runs from different *PAR* and *F* combinations.

Table 1 shows that function f1 gets the best result when the parameter pair (*PAR*, *F*) is (0.9, 0.4). Table S1 shows that function f10 gets the best result when the parameter pair (*PAR*, *F*) is (0.1, 0.6). Table S2 shows that function f21 gets the best result when the parameter pair (*PAR*, *F*) is (0.9, 0.3). Table S3 shows that function f28 gets the best result when the parameter pair (*PAR*, *F*) is (0.7, 0.6). At the same time, it is easy to see that the performance of algorithm varies greatly with different parameter pairs. For example, the result of algorithm varies from 1.35×10^7 with (*PAR*, *F*) being (0.8, 0.9) to 3.53×10^3 with (*PAR*, *F*) being (0.9, 0.4) for Function 1. The result of the algorithm varies from 8.66×10^5 with (*PAR*, *F*) being (0.1, 0.3) to 8.89×10 with (*PAR*, *F*) being (0.9, 0.3) for Function 2. This tells us that different functions have different sensitivities to the parameter *PAR* or *F*.

Table 1. Fitness of f1 for different parameters (*PAR*, *F*).

PAR/F	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.1	1.28×10^7	1.35×10^7	1.17×10^7	7.01×10^6	8.09×10^6	2.08×10^6	1.10×10^6	1.02×10^6	1.17×10^6
0.2	7.20×10^6	7.11×10^6	5.56×10^6	1.27×10^6	8.57×10^5	7.69×10^5	7.01×10^5	6.36×10^5	7.72×10^5
0.3	8.54×10^6	6.17×10^6	2.11×10^6	5.26×10^5	4.86×10^5	5.88×10^5	6.94×10^5	8.08×10^5	1.05×10^6
0.4	5.22×10^6	4.00×10^6	7.65×10^5	3.86×10^5	3.81×10^5	6.83×10^5	1.06×10^6	1.48×10^6	1.63×10^6
0.5	5.69×10^6	2.16×10^6	3.99×10^5	2.53×10^5	4.68×10^5	1.02×10^6	1.65×10^6	2.20×10^6	2.74×10^6
0.6	3.37×10^6	7.32×10^5	1.80×10^5	3.31×10^5	7.86×10^5	1.43×10^6	2.47×10^6	4.55×10^6	4.94×10^6
0.7	2.87×10^6	4.47×10^5	1.14×10^5	3.51×10^5	1.03×10^6	2.52×10^6	5.63×10^6	9.59×10^6	1.52×10^7
0.8	2.92×10^6	2.97×10^5	5.14×10^4	1.19×10^5	5.89×10^5	2.20×10^6	6.67×10^6	2.61×10^7	3.10×10^7
0.9	2.75×10^6	2.10×10^5	1.12×10^4	$\mathbf{3.53 \times 10^3}$	5.00×10^4	5.17×10^5	2.29×10^6	1.03×10^7	2.14×10^7

Observed from the comparison analysis of different parameter pairs on (*PAR*, *F*), the performance of the algorithm is sensitive to the parameter pair (*PAR*, *F*) to different problems. Simultaneously, it demonstrates that a certain parameter adaption scheme is necessary for problem solving. The algorithm aHSDE can obtain the best parameter pair of (*PAR*, *F*) and converge the best solution with the adaptive strategy. This scheme can reduce the dependence on the parameter for the algorithm.

Thus, it can be said that Table 1 and Tables S1–S3 fully demonstrate the effects of the adaptive strategy. In conclusion, the aHSDE is highly successful with the tuned parameter settings of *PAR* and *F* through the learning period and the weighted Lehmer mean method.

5. Experimental Comparison with HS Variants and Well-Known EAs

5.1. aHSDE vs. HS Variants

The experimental results of five algorithms (HS, IHS, SGHS, IGHS and the aHSDE) are reported and compared in Tables S4–S6 with different dimension sizes 10, 50 and 100, respectively. The items "Best", "Mean" and "SD" represent the best and average results and the standard deviation of multiple final results, which are collected in 30 independent runs for each algorithm on each function. Meanwhile, the fitness error is assigned to zero if it was less than 1×10^{-8}.

It can be seen from Tables S4–S6 which can be found in the Supplementary data due to space and readability reasons) that the aHSDE has significantly competitive performance when compared with the canonical HS algorithm and several state-of-the-art HS variants. These data are the statistical results of 30 independent runs with the CEC 2014 benchmark for the 10-, 50- and 100-dimension sizes. In Tables S4–S6, the aHSDE always performs best among its competitors on f1–f3 unimodal functions. Secondly, the performance advantage of the aHSDE to its competitors increases as the dimension increases on f4–f16 multimodal functions. Let us take the concrete example as an illustration of algorithmic performance difference. For example, the mean results of Function 8 of HS, IHS, SGHS and the aHSDE. The IGHS indicates that three of four variants obtain the true optimal solution with the dimension as 10. The mean item of the aHSDE is 7.41×10^{-8}, however, the best mean item of the other three algorithms is 2.52×10^{-2} for Function 8 with the dimension as 50. The mean item of the aHSDE is 2.90×10^{-6}, however, the best mean item of other three algorithms is 1.74×10^{-0} for Function 8 with the dimension as 100. This concrete example indicates that the performance advantage of the aHSDE to its competitors becomes more and more obvious with the increase of dimension size.

Moreover, the performance of the aHSDE is also all better than those of the other four HS variants on f17–f22 hybrid functions, except for Function 19, which has slightly worse performance with a dimension size of 100. Subsequently, Tables S4–S6 indicate that the performance advantage of the aHSDE is not as obvious as the previous benchmark. It performs slightly better than other competitors on the composition functions f23–f30. However, the overall statistical Table 2 tells us that the aHSDE still has the best cases for all the composition functions f23–f30 for all the dimensional sizes. These statistical

experimental comparisons and results analyses indicate that the improvement strategies of the aHSDE have significant impact on performance and its ability on global exploration and local exploitation.

5.2. Overall Statistical Comparison among HS Variants

Table 2 presents the overall statistical comparison results for the aHSDE and its competitors based on the Wilcoxon rank–sum test with the significance level α of 0.05 for each dimension case on all the benchmark functions. The symbols "+", "−" and "~" mean that the aHSDE performs significantly better, significantly worse, or not significantly different compared with its competitors. Overall, it demonstrated that the performance of the aHSDE is quite competitive compared with four HS variants on the CEC2014 benchmark.

Table 2. Overall statistical comparison among aHSDE and HS, improved harmony search (IHS), self-adaptive global-best harmony search (SGHS) and IGHS on CEC2014.

Groups	aHSDE vs	HS(DIM =)			IHS(DIM =)			SGHS(DIM =)			IGHS(DIM =)		
		10	50	100	10	50	100	10	50	100	10	50	100
3 Unimodal Functions	+	3	3	3	3	2	3	3	2	3	3	3	3
	−	0	0	0	0	0	0	0	0	0	0	0	0
	~	0	0	0	0	1	0	0	1	0	0	0	0
13 Simple Multimodal Functions	+	6	11	12	5	11	13	8	11	10	9	13	13
	−	4	1	1	6	1	0	1	0	0	0	0	0
	~	3	1	0	2	1	0	4	2	3	4	0	0
6 Hybrid Functions	+	4	6	6	4	6	5	5	6	5	6	6	6
	−	0	0	0	1	0	0	0	0	0	0	0	0
	~	2	0	0	1	0	1	1	0	1	0	0	0
8 Composition Functions	+	4	6	6	4	5	6	6	4	6	4	6	7
	−	2	1	2	1	2	2	1	3	2	3	0	0
	~	2	1	0	3	1	0	1	1	0	1	2	1
30 All Functions	+	17	26	27	16	24	27	22	23	24	22	28	29
	−	6	2	3	8	3	2	2	3	2	3	0	0
	~	7	2	0	6	3	1	6	4	4	5	2	1

The following facts can be observed from Table 2. For three unimodal functions, the aHSDE outperforms all its competitors for 10, 50 and 100 dimensions. For thirteen multimodal functions, the aHSDE performs a little better than HS on 10 dimensions and performs much better than HS with the increase of dimension size for all competitions and all functions. For six hybrid functions, the aHSDE clearly outperforms HS, IHS, SGHS and IGHS for all the cases. These results illustrate that the aHSDE has a superior advantage to the state-of-the-art HS variants when solving various optimization problems, whose varieties may have no features. For the eight composition functions, the aHSDE significantly outperforms HS, IHS, SGHS and IGHS for 10, 50, 100 dimensions, except that the aHSDE performs comparably to SGHS for the 50-dimensional case and IGHS for the 10-dimensional case, respectively. The advantages are more obvious on higher dimensional functions. As a whole, the aHSDE performs much better than the canonical HS algorithm and HS variants in total on 10, 50, 100 dimensions on 30 benchmark functions for all dimensional cases.

5.3. Comparison with Other Well-Known EAs

In this subsection, the proposed aHSDE algorithm is compared with other state-of-the-art evolutionary algorithms (non harmony ones), including Adaptive Particle Swarm Optimization (APSO) [43] and Evolution Strategy with Covariance Matrix Adaptation (CMA-ES) [44]. The experimental results (mean best and standard deviation of multiple runs) of different algorithms are all collected with the maximum function evaluation number DIM * 10000 respectively, all of which are summarized in Table 3. The best "mean" result for the same function is highlighted in bold.

Table 3. Comparison on mean best and standard deviation of multiple runs of Adaptive Particle Swarm Optimization (APSO), Evolution Strategy with Covariance Matrix Adaptation (CMA-ES) and the aHSDE based on IEEE CEC 2014 benchmarks.

Function	APSO	CMA-ES	aHSDE	Function	APSO	CMA-ES	aHSDE
f1	$2.69 \times 10^{9} \pm 3.28 \times 10^{8}$	$\mathbf{9.42} \times 10^{4} \pm 7.88 \times 10^{4}$	$2.06 \times 10^{5} \pm 9.12 \times 10^{4}$	f16	$1.42 \times 10 \pm 2.37 \times 10^{-1}$	$\mathbf{1.38} \times 10 \pm 5.31 \times 10^{-1}$	$1.78 \times 10 \pm 1.00 \times 10^{0}$
f2	$1.02 \times 10^{11} \pm 2.29 \times 10^{9}$	$2.55 \times 10 \pm 3.85 \times 10^{9}$	$\mathbf{5.77 \times 10^{3}} \pm 6.41 \times 10^{3}$	f17	$2.86 \times 10^{8} \pm 1.28 \times 10^{8}$	$5.49 \times 10^{3} \pm 3.62 \times 10^{3}$	$\mathbf{2.66 \times 10^{3}} \pm 1.83 \times 10^{3}$
f3	$1.19 \times 10^{6} \pm 1.25 \times 10^{6}$	$1.45 \times 10^{4} \pm 5.66 \times 10^{3}$	$\mathbf{1.25 \times 10^{0}} \pm 1.61 \times 10^{0}$	f18	$8.75 \times 10^{9} \pm 3.11 \times 10^{9}$	$1.52 \times 10^{9} \pm 3.93 \times 10^{8}$	$\mathbf{8.88 \times 10} \pm 3.11 \times 10$
f4	$2.49 \times 10^{4} \pm 1.54 \times 10^{3}$	$\mathbf{2.00 \times 10} \pm 2.63 \times 10^{-5}$	$4.43 \times 10 \pm 3.69 \times 10$	f19	$8.45 \times 10^{2} \pm 1.15 \times 10^{2}$	$2.98 \times 10^{2} \pm 4.25 \times 10$	$\mathbf{1.16 \times 10} \pm 1.42 \times 10^{0}$
f5	$2.13 \times 10 \pm 5.61 \times 10^{-2}$	$2.08 \times 10 \pm 6.69 \times 10^{-2}$	$\mathbf{2.01 \times 10} \pm 4.08 \times 10^{-2}$	f20	$1.59 \times 10^{7} \pm 1.37 \times 10^{7}$	$4.61 \times 10^{3} \pm 3.88 \times 10^{3}$	$\mathbf{4.07} \times 10 \pm 9.82 \times 10^{0}$
f6	$4.80 \times 10 \pm 1.79 \times 10^{0}$	$4.09 \times 10^{3} \pm 2.13 \times 10^{0}$	$\mathbf{2.02 \times 10} \pm 3.77 \times 10^{0}$	f21	$1.33 \times 10^{8} \pm 7.50 \times 10^{7}$	$6.86 \times 10^{3} \pm 2.76 \times 10^{3}$	$\mathbf{8.35} \times 10^{2} \pm 2.36 \times 10^{2}$
f7	$1.06 \times 10^{3} \pm 3.85 \times 10$	$2.31 \times 10^{2} \pm 2.83 \times 10$	$\mathbf{2.14 \times 10^{-3}} \pm 4.28 \times 10^{-3}$	f22	$1.31 \times 10^{4} \pm 9.38 \times 10^{3}$	$1.61 \times 10^{3} \pm 2.92 \times 10^{2}$	$\mathbf{8.30 \times 10^{2}} \pm 2.96 \times 10^{2}$
f8	$5.03 \times 10^{2} \pm 3.02 \times 10$	$2.83 \times 10^{2} \pm 2.21 \times 10$	$\mathbf{7.41} \times 10^{-8} \pm 1.94 \times 10^{-8}$	f23	$\mathbf{2.00 \times 10^{2}} \pm 0.00 \times 10^{0}$	$5.79 \times 10^{2} \pm 4.94 \times 10$	$3.44 \times 10^{2} \pm 0.00 \times 10^{0}$
f9	$4.78 \times 10^{2} \pm 6.30 \times 10^{0}$	$3.28 \times 10^{2} \pm 7.65 \times 10$	$\mathbf{7.89 \times 10} \pm 1.80 \times 10$	f24	$\mathbf{2.00 \times 10^{2}} \pm 0.00 \times 10^{0}$	$2.12 \times 10^{2} \pm 7.49 \times 10^{0}$	$2.69 \times 10^{2} \pm 6.50 \times 10^{0}$
f10	$9.30 \times 10^{3} \pm 5.68 \times 10^{2}$	$2.61 \times 10^{2} \pm 1.06 \times 10^{2}$	$\mathbf{1.94 \times 10^{-1}} \pm 4.50 \times 10^{-2}$	f25	$\mathbf{2.00 \times 10^{2}} \pm 0.00 \times 10^{0}$	$2.12 \times 10^{2} \pm 2.97 \times 10^{0}$	$2.07 \times 10^{2} \pm 2.04 \times 10^{0}$
f11	$9.24 \times 10^{3} \pm 4.86 \times 10^{2}$	$\mathbf{1.69 \times 10^{2}} \pm 1.98 \times 10^{2}$	$4.71 \times 10^{3} \pm 5.61 \times 10^{2}$	f26	$1.86 \times 10^{2} \pm 2.68 \times 10$	$1.25 \times 10^{-2} \pm 5.51 \times 10^{-1}$	$\mathbf{1.00 \times 10^{2}} \pm 6.01 \times 10^{-2}$
f12	$5.91 \times 10^{0} \pm 1.32\times 10^{0}$	$3.03 \times 10^{-1} \pm 2.18 \times 10^{0}$	$\mathbf{9.57 \times 10^{-2}} \pm 4.16 \times 10^{-2}$	f27	$\mathbf{2.00 \times 10^{2}} \pm 0.00 \times 10^{0}$	$\mathbf{1.07 \times 10^{3}} \pm 2.30 \times 10^{2}$	$8.76 \times 10^{2} \pm 1.26 \times 10^{2}$
f13	$1.03 \times 10 \pm 7.53 \times 10^{-1}$	$5.51 \times 10^{0} \pm 3.07 \times 10^{-1}$	$\mathbf{3.31 \times 10^{-1}} \pm 6.32 \times 10^{-2}$	f28	$\mathbf{2.00} \times 10^{2} \pm 0.00 \times 10^{0}$	$2.79 \times 10^{3} \pm 5.92 \times 10^{2}$	$1.28 \times 10^{3} \pm 8.88 \times 10$
f14	$3.95 \times 10^{2} \pm 2.22 \times 10$	$7.53 \times 10 \pm 8.08 \times 10^{0}$	$\mathbf{3.30 \times 10^{-1}} \pm 1.12 \times 10^{-1}$	f29	$\mathbf{2.00 \times 10^{2}} \pm 0.00 \times 10^{0}$	$3.52 \times 10^{4} \pm 5.34 \times 10^{3}$	$2.35 \times 10^{7} \pm 1.69 \times 10^{7}$
f15	$1.05 \times 10^{6} \pm 0.00 \times 10^{0}$	$1.02 \times 10^{4} \pm 3.24 \times 10^{4}$	$\mathbf{8.09 \times 10^{0}} \pm 2.32 \times 10^{0}$	f30	$\mathbf{2.00 \times 10^{2}} \pm 0.00 \times 10^{0}$	$6.48 \times 10^{5} \pm 1.31 \times 10^{5}$	$8.93 \times 10^{3} \pm 6.76 \times 10^{2}$

Observed in Table 3, it can be seen that APSO, CMA-ES and the aHSDE perform best on 7, 4 and 19 benchmarks respectively from the 30 benchmark functions. It should be further noted that APSO outperforms the aHSDE on eight problems and CMA-ES outperforms the aHSDE on six functions among this IEEE CEC 2014 benchmark suite. Therefore, generally speaking, the aHSDE significantly outperforms APSO and CMA-ES on most of the benchmark functions. However, it should be especially noted that APSO outperforms CMA-ES and the aHSDE on the composition functions, which indicates that APSO is promising for the composition, or the highly complex problems. Comparatively speaking, the aHSDE has better overall performance on multiple types of problems.

6. Conclusions

Based on the analysis of HSA and the knowledge and experience of the musician, a new adaptive harmony search algorithm is proposed (aHSDE) for global optimization in this paper. It enhances the performance of HS with a differential mutation for the pitch adjustment of *HS*, the mechanism of decreasing *HMS* linearly, and the parameter adaptation of *PAR* and *F*. Firstly, the mutual influence and cooperation of three strategies and key parameters on the aHSDE are analyzed and verified in detail. After this, the performance of the aHSDE is comprehensively evaluated on IEEE CEC 2014 Benchmarks with 10-, 50- and 100-dimension sizes. The experimental results indicate that the aHSDE outperforms the canonical HS algorithm and three advanced HS variants, including IHS, SGHS and IGHS. Furthermore, other state-of-the-art metaheuristic algorithms, namely APSO and CMA-ES, are also used as competitors to evaluate the aHSDE.

Supplementary Materials: The following are available online at http://www.mdpi.com/2076-3417/10/8/2916/s1, Table S1: Fitness of f10 for different parameters (PAR, F). Table S2: Fitness of f21 for different parameters (*PAR*, *F*). Table S3 : Fitness of f28 for different parameters (*PAR*, *F*). Table S4: Performance comparison among five harmony search algorithms for f1–f30 (DIM = 10). Table S5: Performance comparison among five harmony search algorithms for f1–f30 (DIM = 50). Table S6: Performance comparison among five harmony search algorithms for f1–f30 (DIM = 100).

Author Contributions: Conceptualization, X.Z.; methodology, X.Z., R.L.; investigation, J.H.; data curation, R.L.; writing—original draft preparation, Z.L.; writing—review and editing, J.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Beijing Natural Science Foundation, grant number 1202020 and National Natural Science Foundation of China, grant number 61973042, 71772060. The APC was funded by Xinchao Zhao with his funds.

Acknowledgments: We will also express our awfully thanks to the Swarm Intelligence Research Team of BeiYou University.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Geem, Z.W.; Kim, J.H.; Loganathan, G. A New Heuristic Optimization Algorithm: Harmony Search. *Simulation* **2001**, *76*, 60–68. [CrossRef]
2. Geem, Z.W. Optimal cost design of water distribution networks using harmony search. *Eng. Optim.* **2006**, *38*, 259–277. [CrossRef]
3. Mahdavi, M.; Fesanghary, M.; Damangir, E. An improved harmony search algorithm for solving optimization problems. *Appl. Math. Comput.* **2007**, *188*, 1567–1579. [CrossRef]
4. Pan, Q.-K.; Suganthan, P.N.; Tasgetiren, M.F.; Liang, J. A self-adaptive global best harmony search algorithm for continuous optimization problems. *Appl. Math. Comput.* **2010**, *216*, 830–848. [CrossRef]
5. Zhao, F.; Liu, Y.; Zhang, C.; Wang, J. A self-adaptive harmony PSO search algorithm and its performance analysis. *Expert Syst. Appl.* **2015**, *42*, 7436–7455. [CrossRef]
6. Valian, E.; Tavakoli, S.; Mohanna, S. An intelligent global harmony search approach to continuous optimization problems. *Appl. Math. Comput.* **2014**, *232*, 670–684. [CrossRef]
7. Luo, K.; Ma, J.; Zhao, Q. Enhanced self-adaptive global-best harmony search without any extra statistic and external archive. *Inf. Sci.* **2019**, *482*, 228–247. [CrossRef]
8. Geem, Z.W.; Sim, K.-B. Parameter-setting-free harmony search algorithm. *Appl. Math. Comput.* **2010**, *217*, 3881–3889. [CrossRef]
9. Ouyang, H.-B.; Gao, L.-Q.; Li, S.; Kong, X.-Y.; Wang, Q.; Zou, D.-X. Improved Harmony Search Algorithm: LHS. *Appl. Soft Comput.* **2017**, *53*, 133–167. [CrossRef]
10. Assad, A.; Deep, K. A Hybrid Harmony search and Simulated Annealing algorithm for continuous optimization. *Inf. Sci.* **2018**, *450*, 246–266. [CrossRef]
11. Zhu, Q.; Tang, X.; Li, Y.; Yeboah, M.O. An improved differential-based harmony search algorithm with linear dynamic domain. *Knowl.-Based Syst.* **2020**, *187*, 104809. [CrossRef]
12. Zhang, T.; Geem, Z.W. Review of harmony search with respect to algorithm structure. *Swarm Evol. Comput.* **2019**, *48*, 31–43. [CrossRef]
13. Saka, M.; Hasançebi, O.; Geem, Z.W. Metaheuristics in structural optimization and discussions on harmony search algorithm. *Swarm Evol. Comput.* **2016**, *28*, 88–97. [CrossRef]
14. Geem, Z.W. Novel derivative of harmony search algorithm for discrete design variables. *Appl. Math. Comput.* **2008**, *199*, 223–230. [CrossRef]
15. Couckuyt, I.; Deschrijver, D.; Dhaene, T. Fast calculation of multiobjective probability of improvement and expected improvement criteria for Pareto optimization. *J. Glob. Optim.* **2013**, *60*, 575–594. [CrossRef]
16. Manjarrés, D.; Landa-Torres, I.; Gil-Lopez, S.; Del Ser, J.; Bilbao, M.; Salcedo-Sanz, S.; Geem, Z.W. A survey on applications of the harmony search algorithm. *Eng. Appl. Artif. Intell.* **2013**, *26*, 1818–1831. [CrossRef]
17. Ertenlice, O.; Kalayci, C.B.; Kalaycı, C.B. A survey of swarm intelligence for portfolio optimization: Algorithms and applications. *Swarm Evol. Comput.* **2018**, *39*, 36–52. [CrossRef]

18. Geem, Z.W.; Kim, J.; Loganathan, G. Harmony Search Optimization: Application to Pipe Network Design. *Int. J. Model. Simul.* **2002**, *22*, 125–133. [CrossRef]
19. Zhao, X.; Liu, Z.; Hao, J.; Li, R.; Zuo, X. Semi-self-adaptive harmony search algorithm. *Nat. Comput.* **2017**, *16*, 619–636. [CrossRef]
20. Yi, J.; Gao, L.; Li, X.; Shoemaker, C.A.; Lu, C. An on-line variable-fidelity surrogate-assisted harmony search algorithm with multi-level screening strategy for expensive engineering design optimization. *Knowl.-Based Syst.* **2019**, *170*, 1–19. [CrossRef]
21. Vasebi, A.; Fesanghary, M.; Bathaee, S. Combined heat and power economic dispatch by harmony search algorithm. *Int. J. Electr. Power Energy Syst.* **2007**, *29*, 713–719. [CrossRef]
22. Geem, Z.W. Optimal Scheduling of Multiple Dam System Using Harmony Search Algorithm. In Proceedings of the International Work Conference on Artificial Neural Networks, San Sebastin, Spain, 20−22 June 2007; LNCS 4507; pp. 316–323.
23. Geem, Z.W. Harmony search optimization to the pump-included water distribution network design. *Civ. Eng. Environ. Syst.* **2009**, *26*, 211–221. [CrossRef]
24. Geem, Z.W. Particle-swarm harmony search for water network design. *Eng. Optim.* **2009**, *41*, 297–311. [CrossRef]
25. Lin, Q.; Chen, J. A novel micro-population immune multiobjective optimization algorithm. *Comput. Oper. Res.* **2013**, *40*, 1590–1601. [CrossRef]
26. Manjarres, D.; Ser, J.D.; Gil-Lopez, S.; Vecchio, M.; Landa-Torres, I.; Lopez-Valcarce, R. A novel heuristic approach for distance-and connectivity-based multi-hop node localization in wireless sensor networks. *Soft Comput.* **2013**, *17*, 17–28. [CrossRef]
27. Landa-Torres, I.; Gil-Lopez, S.; Ser, J.D.; Salcedo-Sanz, S.; Manjarres, D.; Portilla-Figueras, J.A. Efficient citywide planning of open WiFi access networks using novel grouping harmony search heuristics. *Eng. Appl. Artif. Intell.* **2013**, *26*, 1124–1130. [CrossRef]
28. Peng, Z.-R.; Yin, H.; Dong, H.-T.; Li, H.; Pan, A. A Harmony Search Based Low-Delay and Low-Energy Wireless Sensor Network. *Int. J. Future Gener. Commun. Netw.* **2015**, *8*, 21–32. [CrossRef]
29. Mohsen, A. A Robust Harmony Search Algorithm Based Markov Model for Node Deployment in Hybrid Wireless Sensor Networks. *Int. J. Geomate* **2016**, *11*. [CrossRef]
30. Nikravan, M. Combining Harmony Search and Learning Automata for Topology Control in Wireless Sensor Networks. *Int. J. Wirel. Mob. Netw.* **2012**, *4*, 87–98. [CrossRef]
31. Degertekin, S.O. Improved harmony search algorithms for sizing optimization of truss structures. *Comput. Struct.* **2012**, *92*, 229–241. [CrossRef]
32. Kim, Y.-H.; Yoon, Y.; Geem, Z.W. A comparison study of harmony search and genetic algorithm for the max-cut problem. *Swarm Evol. Comput.* **2019**, *44*, 130–135. [CrossRef]
33. Boryczka, U.; Szwarc, K. The Harmony Search algorithm with additional improvement of harmony memory for Asymmetric Traveling Salesman Problem. *Expert Syst. Appl.* **2019**, *122*, 43–53. [CrossRef]
34. Seyedhosseini, S.M.; Esfahani, M.J.; Ghaffari, M. A novel hybrid algorithm based on a harmony search and artificial bee colony for solving a portfolio optimization problem using a mean-semi variance approach. *J. Cent. South Univ.* **2016**, *23*, 181–188. [CrossRef]
35. Shams, M.; El-Banbi, A.; Sayyouh, H. Harmony search optimization applied to reservoir engineering assisted history matching. *Pet. Explor. Dev.* **2020**, *47*, 154–160. [CrossRef]
36. Lee, K.S.; Geem, Z.W. A new structural optimization method based on the harmony search algorithm. *Comput. Struct.* **2004**, *82*, 781–798. [CrossRef]
37. Price, K.; Storn, R.; Lampinen, J. *Differential Evolution: A Practical Approach to Global Optimization*; Springer Science & Business Media: Berlin, Germany, 2006.
38. Park, S.-Y.; Lee, J.-J. Stochastic Opposition-Based Learning Using a Beta Distribution in Differential Evolution. *IEEE Trans. Cybern.* **2015**, *46*, 2184–2194. [CrossRef]
39. Qin, A.; Forbes, F. Harmony search with differential mutation based pitch adjustment. In Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation-GECCO'11; Association for Computing Machinery (ACM), Dublin, Ireland, 12−16 July 2011; pp. 545–552.
40. Tanabe, R.; Fukunaga, A.S.; Tanabe, R. Improving the search performance of SHADE using linear population size reduction. In Proceedings of the 2014 IEEE Congress on Evolutionary Computation (CEC), Beijing, China, 6–11 July 2014; pp. 1658–1665. [CrossRef]

41. Pant, M.; Zaheer, H.; Garcia-Hernandez, L.; Abraham, A. Differential Evolution: A review of more than two decades of research. *Eng. Appl. Artif. Intell.* **2020**, *90*, 103479. [CrossRef]
42. Peng, F.; Tang, K.; Chen, G.; Yao, X. Multi-start JADE with knowledge transfer for numerical optimization. In Proceedings of the 2009 IEEE Congress on Evolutionary Computation, Trondheim, Norway, 18–21 May 2009; pp. 1889–1895. [CrossRef]
43. Zhan, Z.-H.; Zhang, J.; Li, Y.; Chung, H.S.-H. Adaptive particle swarm optimization. *IEEE Trans. Syst. Man Cybern. Part B (Cybern.)* **2009**, *39*, 1362–1381. [CrossRef]
44. Hansen, N.; Müller, S.D.; Koumoutsakos, P. Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). *Evol. Comput.* **2003**, *11*, 1–18. [CrossRef] [PubMed]
45. Liang, J.J.; Qu, B.Y.; Suganthan, P.N. *Problem Definitions and Evaluation Criteria for the CEC 2014 Special Session and Competition on Single Objective Real-Parameter Numerical Optimization*; Computational Intelligence Laboratory, Zhengzhou University: Zhengzhou, China; Technical Report, Nanyang Technological University: Singapore, 2013.
46. Xie, F.; Butt, M.M.; Li, Z. A feasible flow-based iterative algorithm for the two-level hierarchical time minimization transportation problem. *Comput. Oper. Res.* **2017**, *86*, 124–139. [CrossRef]
47. Singh, K.; Jain, A.; Mittal, A.; Yadav, V.; Singh, A.A.; Jain, A.K.; Gupta, M. Optimum transistor sizing of CMOS logic circuits using logical effort theory and evolutionary algorithms. *Integration* **2018**, *60*, 25–38. [CrossRef]
48. Subashini, M.M.; Sahoo, S.K.; Sunil, V.; Easwaran, S. A non-invasive methodology for the grade identification of astrocytoma using image processing and artificial intelligence techniques. *Expert Syst. Appl.* **2016**, *43*, 186–196. [CrossRef]
49. Shang, R.H.; Dai, K.Y.; Jiao, L.C.; Stolkin, R. Improved memetic algorithm based on route distance grouping for Multi-objective Large Scale Capacitated Arc Routing Problems. *IEEE Trans. Cybern.* **2016**, *6*, 1000–1013. [CrossRef]
50. Li, J.-Q.; Pan, Q.-K.; Tasgetiren, M.F. A discrete artificial bee colony algorithm for the multi-objective flexible job-shop scheduling problem with maintenance activities. *Appl. Math. Model.* **2014**, *38*, 1111–1132. [CrossRef]