algorithms

MDPI

Article

A Reward Population-Based Differential Genetic Harmony Search Algorithm

Yang Zhang [1,*], Jiacheng Li [2] and Lei Li [1]

[1] Department of Applied Informatics, Faculty of Science and Engineering, Hosei University, Tokyo 184-8584, Japan; lilei@hosei.ac.jp
[2] Department of Electrical, Electronics and Information Engineering, Faculty of Engineering, Kanagawa University, Yokohama 221-8686, Japan; lijiacheng@kanagawa-u.ac.jp
* Correspondence: yang.zhang.6j@stu.hosei.ac.jp; Tel.: +81-70-4134-6022

Abstract: To overcome the shortcomings of the harmony search algorithm, such as its slow convergence rate and poor global search ability, a reward population-based differential genetic harmony search algorithm is proposed. In this algorithm, a population is divided into four ordinary sub-populations and one reward sub-population, for each of which the evolution strategy of the differential genetic harmony search is used. After the evolution, the population with the optimal average fitness is combined with the reward population to produce a new reward population. During an experiment, tests were conducted first on determining the value of the harmony memory size (HMS) and the harmony memory consideration rate (HMCR), followed by an analysis of the effect of their values on the performance of the proposed algorithm. Then, six benchmark functions were selected for the experiment, and a comparison was made on the calculation results of the standard harmony memory search algorithm, reward population harmony search algorithm, differential genetic harmony algorithm, and reward population-based differential genetic harmony search algorithm. The result suggests that the reward population-based differential genetic harmony search algorithm has the merits of a strong global search ability, high solving accuracy, and satisfactory stability.

Keywords: harmony search algorithm; reward population; differential evolution algorithm; mutation strategy; genetic algorithm

Citation: Zhang, Y.; Li, J.; Li, L. A Reward Population-Based Differential Genetic Harmony Search Algorithm. *Algorithms* **2022**, *15*, 23. https://doi.org/10.3390/a15010023

Academic Editors: Essam H. Houssein and Frank Werner

Received: 30 November 2021
Accepted: 11 January 2022
Published: 14 January 2022

1. Introduction

With the development of big data, cloud computing, artificial intelligence, and other technologies, the data size of networks has witnessed fast growth, and as a result, it has been more common to solve optimization problems that are similar to traffic networks, such as vehicle route planning, spacecraft design, and wireless sensor layouts [1]. Usually, these optimization problems can be expressed with mathematical programming forms. For general simple optimization problems, mathematical programming and iterative algorithms may be used for complex large optimization problems, however, it is quite difficult to seek a global or approximate optimal solution within a reasonable time with these traditional methods. Therefore, to smoothly solve complex large optimization problems, the heuristic algorithm was proposed and has received increasing attention in recent decades [2].

In computer science and mathematical optimization, a metaheuristic is a higher-level procedure or heuristic designed to find, generate, or select a heuristic that may provide a sufficiently good solution to an optimization problem, especially with incomplete or imperfect information or limited computation capacity [3,4]. The common meta-heuristic algorithms include a genetic algorithm, simulated annealing algorithm, and particle swarm optimization algorithm [5,6]. Unlike meta heuristics, which are independent of the problem, heuristic algorithms depend on a specific problem. Heuristic algorithms are essentially methods by trial and error [7]. In the process of seeking the optimal solution, it can change its search path according to individual or global experience. When it becomes

impossible or difficult to find the optimal solution of the problem, a heuristic algorithm is an efficient method to obtain the feasible solution [8]. Harmony search is a heuristic global search algorithm introduced in 2001 by Zong Woo Geem, Joong Hoon Kim, and G. V. Loganathan [9]. Inspired by the improvisation of musicians, it simulates musicians' improvisation process to achieve subtle harmonies. The HS algorithm is characterized by merits such as a simple structure, less demand for parameters, a high convergence rate, and strong robustness, which facilitates its application in optimizing a system's continuous and discrete variables [10]. We hope that the HS algorithm can effectively solve some optimization problems. In recent years, scholars have constantly improved the algorithm and made some progress [11–14].

Mahdavi et al. used a new method for generating new solution vectors and set the pitch adjustment rate and distance bandwidth as dynamic changes, improving the accuracy and convergence rate of the HS algorithm [15]. Khalili et al. changed all key parameters of the HS algorithm into a dynamic mode without predefining any parameters. This algorithm is referred to as the global dynamic harmony search, and such modification imbued the algorithm with outstanding performance on unimodal functions and multimodal functions [16]. In view of the drawbacks of HS such as a low convergence rate and solving accuracy in solving complex problems, Ouyang et al. proposed an adaptive global modified harmony search (MHS) that fuses local searches [17]. Pan et al. proposed the concepts of the maximum harmony memory consideration rate, the minimum harmony memory consideration rate, the maximum regulation bandwidth, the minimum regulation bandwidth, etc. on the basis of the standard HS algorithm, and they achieved an automatic regulation mechanism for the relevant parameters of the harmony search algorithm by improving the current harmony memory consideration rate for each iteration and changing the bandwidth with the number of iterations [18]. To improve the efficiency of the HS algorithm and make up for its drawback of easily getting into local search, Zhang Kangli et al. improved the generation of the initial solution vector's harmony memory and proposed an improved harmony algorithm, ALHS [19]. Given the low evolution efficiency of single populations and fewer studies on multi-population improvement, S. Z. Zhao combined dynamic multi-swarm particle swarm optimization (DMS-PSO) with the HS algorithm and simplified it into the dynamic multi-swarm particle swarm optimization harmony search (DMS-PSO-HS). These sub-populations are frequently re-combined, and there is information exchange among particles in the entire population. Compared with DMS-PSD and HS, DMS-PSO-HS is improved from the aspect of multi-mode and combinatorial testing [20]. As to the problems with the HS algorithm in solving high-dimensional multi-objective optimization problems, Zhang proposed an improved differential evolved harmony search algorithm. In this algorithm, mutation and crossover are adopted to substitute the original pitch adjustment in the HS optimization algorithm, thus improving the global search ability of the algorithm [21]. To effectively solve integrated process planning and scheduling (IPPS) problems, Wu et al. proposed a nested method for single-objective IPPS problems. On the external layer of the method, the HS algorithm was used to determine the manufacturing feature processing sub-path, and static and dynamic scoring methods were proposed for the sub-path to guide the search on the external layer. Meanwhile, a genetic algorithm was adopted on the internal layer to determine machine allocation and operation series. Upon combining the two algorithms, the validity of the proposed method was proved through a test of benchmark instances [22].

Inspired by the above, the improvement of the HS algorithm mainly focuses on the improvement of parameters and infusion with other algorithms. In this paper, to solve the main problems with the algorithm, such as its poor global search ability and poor solution accuracy, a reward population-based differential harmony search algorithm is proposed. The highlight of the proposed algorithm is as follows. (a) We hope that by using the multiple population reward mechanisms, the population diversity can be increased and the algorithm can avoid falling into local optimization as much as possible. (b) In the step of generating a new harmony, excellent individuals in existing populations are

utilized for optimization, thus improving the convergence rate of the algorithm; at the same time, on the basis of differential evolution and genetic operations, the difference among individuals is utilized flexibly for search guidance, thus improving the search ability of the algorithm. Finally, in a numerical experiment, the validity of the algorithm is verified through comparison.

2. Harmony Search Algorithm

The harmony search algorithm is a novel intelligent optimization algorithm. Like the genetic algorithm, which simulates biological evolution, the simulated annealing algorithm, which simulates physical annealing, and the particle swarm optimization algorithm, which simulates flocks of birds, the harmony algorithm simulates the principle of concert performances [23].

Let us assume that a band consists of n people, and everyone plays a specific musical instrument. Then, all sounds being played correspond to a group of harmonies after combination, which is: $X = x_1, x_2, \ldots, x_n$. As the tone of the initial harmony is not necessarily the best, they need to get the best harmony through continuous cooperation and rehearsal. For convenience of comparison, a mathematical function $f(x)$ can be used to measure whether a harmony sounds good in the whole process. The $f(x)$ serves as a general director, as well as a general judgment criteria. If the requirements are not met, the performance should be constantly adjusted until a set of satisfactory harmonies is achieved. This is the optimization process of the harmony search algorithm.

2.1. Parameters Involved in Harmony Search Algorithm

In short, for the harmony search algorithm (HS), all solution vectors (decision variable sets) are stored in harmony memory (HM). The main parameters of the HS algorithm are the harmony memory size (HMS), harmony memory consideration rate (HMCR), pitch adjusting rate (PAR), distance bandwidth (BW), and number of improvisations or stopping criterion (NI).

Harmony memory size (HMS): As the music of each musical instrument has a specific range, a solution space may be constructed on the basis of such a performance range. Then, the solution space is used to generate a harmony memory randomly. Therefore, the size of the harmony memory should be designated first.

Harmony memory consideration rate (HMCR): In each iteration, a set of harmonies should be extracted from the harmony memory at a specific rate. Upon micro tuning the set, a set of new harmonies is obtained. Then, a judgment is made on whether the new harmonies are better than the worst harmony in the harmony memory. The process of this judgment involves a comparison done on the basis of the function $f(x)$ mentioned above. Therefore, a random harmony memory consideration rate should be generated.

Pitch adjusting rate (PAR): A set of harmonies should be selected from the harmony memory at a specific consideration rate for pitch adjustment.

Tone pitch adjusting bandwidth (BW): As mentioned above, a set of harmonies will be selected from the memory for pitch adjustment at a specific consideration rate. Bandwidth (BW) refers to the adjustment amplitude.

Maximum times of creation (T_{max}): This refers to the times of creation by concert performers, or namely, the number of times the whole harmony adjustment process needs to be repeated.

2.2. Flow of Harmony Search Algorithm

(1) Initialize the problem and algorithm parameters

In general, the global optimization problem can be summarized as follows. Minimize $f(x)$ subject to

$$x_i \in X_i \qquad i = 1, 2, \cdots, N. \tag{1}$$

where, $f(x)$ is the objective function, x is the set of the decision variables x_i, N is the number of decision variables, X_i is the set of possible ranges of values for each decision variable, the upper bound for each decision variable is $UB(i)$, and the lower bound is $LB(i)$; then $LB(i) \leq X_i \leq UB(i)$.

In Section 2.1, it was introduced that five parameters are involved in harmony research. At the very beginning of the algorithm, these values should be set properly.

(2) Initialize harmony memory

A harmony memory with a size of HMS is generated on the basis of the solution space, as follows.

$$HM = \begin{bmatrix} x_1^1 & x_2^1 & \cdots & x_{N-1}^1 & x_N^1 \\ x_1^2 & x_2^2 & \cdots & x_{N-1}^2 & x_N^2 \\ \cdot & \cdot & \cdots & \cdot & \cdot \\ \cdot & \cdot & \cdots & \cdot & \cdot \\ \cdot & \cdot & \cdots & \cdot & \cdot \\ x_1^{HMS-1} & x_2^{HMS-1} & \cdots & x_{n-1}^{HMS-1} & x_N^{HMS-1} \\ x_1^{HMS} & x_2^{HMS} & \cdots & x_{N-1}^{HMS} & x_N^{HMS} \end{bmatrix} \tag{2}$$

Each decision variable is generated as follows: $x_i^j = LB(i) + (UB(i) - LB(i)) * r$ for $i = 1, 2, \cdots, N$ and $j = 1, 2, \cdots HMS$, where r is a random number between 0 and 1.

(3) Generation of new harmony

A new harmony vector is generated by three rules: (a) memory consideration, (b) pitch adjustment, and (c) random selection [24]. First, randomly generate a random number r_1 in the range [0, 1] and compare r_1 with the initialized HMCR. If $r_1 < HMCR$, take a variable randomly from the initialized HM, which is called memory consideration. Otherwise, it can be obtained by random selection (i.e., randomly generated between the search boundaries). In the last step, take a new harmony variable. If it is updated by the memory consideration, it needs to be adjusted, and a variable r_2 between [0,1] is randomly generated. If $r_2 < PAR$, adjust the variable on the basis of the initialized BW and get a new variable, which is called pitch adjustment. The pitch adjustment rule is given as:

$$x'^{new}_i = x_i^{new} \pm r * BW \tag{3}$$

where r is a random number between 0 and 1.

(4) Update of harmony memory

The newly obtained harmony is calculated with $f(x)$. If the new harmony has a better objective function solution than the poorest solution in the above initialized harmony memory (HM), the new harmony will substitute the poorest harmony in the harmony memory.

(5) Judgment on termination

Check whether current times of creation have reached the above initialized maximum times T_{max} of creation. If no, Steps 3–4 should be repeated until the maximum times of creation are reached.

The complete steps of the harmony search algorithm are shown in Algorithm 1 .

Algorithm 1: HS Algorithm

Initialize the algorithm parameters HMS, HMCR, PAR, BW, T_{max}
Initialize the harmony memory.
Repeat
  Improvise a new harmony as:
  for all i, do
    $x_i^{new} \rightarrow \begin{cases} \text{memory consideration with probability HMCR} \\ \text{random selection with probability 1-HMCR} \end{cases}$
    $if \ \ x_i^{new} \in$HM, then
      $x'^{new}_i = \begin{cases} x_i^{new} \pm r * BW & \text{with probability PAR} \\ x_i^{new} & \text{with probability 1-PAR} \end{cases}$
    end if
  end for
  if the new harmony vector is better than the worst
one in the original HM, then
  update harmony memory
  end if
Until T_{max} is fulfilled
Return best harmony

3. Improved Harmony Search Algorithm

Although the harmony search algorithm has the merits of satisfactory realizability, there are still some drawbacks to be overcome, such as a poor global search ability, inadequate accuracy in dealing with high dimensional optimization problems, and difficulties in pitch adjustment [25]. In this paper, a reward population-based differential genetic harmony search algorithm (RDEGA-HS) is proposed. Furthermore, the reward population harmony search algorithm (RHS) and differential genetic harmony search algorithm (DEGA-HS) are designed to facilitate comparison of the results in the experiment.

3.1. Reward Population Harmony Search Algorithm (RHS)

In this paper, we propose a multi-population model based on reward population. First, the initialized harmony memory is divided into multiple sub-memories, including several common sub-memories and one reward memory. The size of each memory is subject to the parameter λ. Namely,

$$Popsize_i = Popsize * \lambda_i \qquad i = 1, 2, \cdots, N. \tag{4}$$

$$Pop = \cup Pop_i \tag{5}$$

$$\sum \lambda_i = 1 \tag{6}$$

where, Pop and Pop_i refer to the total harmony memories and sub-harmony memories, respectively, $Popsize$ and $Popsize_i$ refer to the number of the total harmony memories and sub-harmony memories, respectively, λ_i refers to the proportion of each sub-harmony memory, and i refers to the number of sub-harmony memories.

Each harmony memory evolves separately. Furthermore, after each evolution, the sub-harmony memory with the optimal average fitness is combined with the reward harmony memory to form a new reward harmony memory. At the same time, poor harmony factors are eliminated to guarantee stability in the number of newly generated harmony memories. In this paper, four sub-harmony memories and one reward memory are set.

3.2. Differential Genetic Harmony Search Algorithm (DEGA-HS)

3.2.1. Mutation of Differential Evolution

The differential evolution algorithm (DE) is a population-based heuristic search algorithm [26]. The flows of evolution are quite similar to those of the genetic algorithm.

The operators of both are selection, crossover, and mutation. For the genetic algorithm, crossover goes before mutation. For the DE algorithm, however, mutation goes before crossover. Furthermore, with regard to the specific definition of these operations, there is a difference between the two algorithms.

Let us assume that a population consists of NP N-dimensional vector individuals. Then, each population can be expressed as: $Pop = (x_1, x_2, \cdots, x_{NP})$. Generally, after the initialization of a population, the differential mutation operation will generate mutation vectors. The mutation strategy of the DE algorithm is advantageous in overcoming premature convergence. Therefore, it was considered that the mutation operation of the DE algorithm be introduced into the harmony search algorithm to overcome the premature convergence of the harmony search algorithm, thus improving the convergence rate.

There are many forms of mutation operations, where the common forms are DE/rand/1/bin, DE/best/1/bin, DE/rand to best/1/bin, DE/best/2/bin and so on. As to the representative methods, they are "selection of DE algorithm", "basic vector selection method", "number of differential vectors", and "crossover methods". Generally, they are expressed in the form of DE/base/num/cross. If "best" is selected for the base part, it means the optimal individual from the population is selected as the basic vector; if "rand" is selected, it means random selection. If "bin" is selected for the cross part, it means binomial crossover is adopted. The results show that if "rand" is selected in the base part, it is favorable for maintaining diversity in populations; if "best" is selected, it stresses the convergence rate [27].

For each individual x_i in a population, a mutation vector of the differential evolution algorithm is generated as per the following formula.

$$v_i = x_{r1} + F(x_{r2} - x_{r3}) \tag{7}$$

r_1, r_2, and r_3 are selected randomly and differ from each other. x_{r1} is referred to as a basic vector, $(x_{r2} - x_{r3})$ is referred to as a differential vector, and F is a scaling factor, $F \in (0, 1)$. F is used to control the size of the differential vector. It is worth noting that an excessively great value of F will cause difficulties in the convergence of populations, and an excessively small value will reduce the convergence speed.

In this paper, two mutation strategies of DE are fused to improve the proposed algorithm. "DE/rand/bin" is selected randomly during iteration, without using the current optimal solution. Therefore, the convergence rate is relatively low, but the global search ability is strong. During iteration, as "DE/best/1/bin" utilizes the current optimal solution, the algorithm is able to better improve the convergence rate. On this basis, "DE/best/1/bin" is flexibly used in the memory consideration of the HS algorithm, while "DE/rand/1/bin" is used for substituting pitch adjustment in the HS algorithm. The two mutation generation strategies are as follows.

$$\text{DE/best/1/bin} \qquad v_i = x_{best} + F(x_{r1} - x_{r2}) \tag{8}$$

$$\text{DE/rand/1/bin} \qquad v_i = x_{rand} + F(x_{r1} - x_{r2}) \tag{9}$$

3.2.2. Partheno-Genetic Algorithm

To increase the diversity of harmony factors, the Partheno-genetic algorithm (PGA) was considered to increase the means of generating new harmonies. For individuals with high randomness, they are subject to a shift operation, inversion operation, and permutation operation, and thus, three new harmonies are obtained. Then, the harmony with the optimal fitness is selected as the new individual introduced by the genetic operation.

The PGA is an improved GA that was put forward by Li and Tong in 1999 [28]. PGA is a genetic method for reproduction through gene shift, gene permutation, and gene inversion. Next, we will introduce genetic operators, namely, the permutation operator, shift operator, and inversion operator, which are characterized by a simple genetic operation,

decreasing the requirement for diversity in the initial population and avoiding "premature convergence" [29].

(1) Permutation Operator.

The permutation operator is a process where genes at specific locations of a chromosome are exchanged at a certain probability and the locations of the exchanged genes are random. Permutation operators may be divided into single-point permutation and multi-point permutation. As to single-point permutation, two genes on a chromosome are exchanged, and in multi-point permutation, multiple genes on a chromosome are exchanged. The process of the two permutation operators is shown in Figure 1, where B and B' are the single-point permutation and multi-point permutation result for Chromosome A, respectively.

$$\begin{aligned} A &= (c_1, c_2, c_3, \cdots, c_{i-1}, c_i, c_{i+1}, \cdots, c_{j-1} c_j, c_{j+1}, \cdots, c_N) \\ &\Downarrow \\ B &= (c_1, c_2, c_3, \cdots, c_{i-1}, c_j, c_{i+1}, \cdots, c_{j-1}, c_i, c_{j+1}, \cdots, c_N) \\ B' &= (c_1, c_2, c_3, \cdots, c_{i-1}, c_j, c_{i+1}, \cdots, c_N, c_i, c_{j+1}, \ldots, c_{j-1}) \end{aligned}$$

Figure 1. Permutation operator operation.

(2) Shift Operator.

The shift operator is a process in which genes in some substrings of a chromosome are shifted backward successively, while the last gene in the substring is shifted to the foremost location. The substrings whose genes are shifted in a chromosome and their length are selected randomly. The gene shift can be divided into single-point shift and multi-point shift. As to single-point shift, only one substring of a chromosome is selected for gene shift, and in multi-point shift, multiple substrings of a chromosome are selected for gene shift. The process of the shift operation is shown in Figure 2. H is a chromosome including multiple genes $k_i (i \in 1, 2, \cdots N)$. I and I' are chromosomes obtained by implementing single-point shift and multi-point shift operations for H, respectively.

$$\begin{aligned} H &= (k_1, k_2, k_3, k_4, k_5, \cdots, k_{i-2}, k_{i-1} k_i, k_{i+1}, k_{i+2}, \cdots, k_N) \\ &\Downarrow \\ I &= (k_4, k_1, k_2, k_3, k_5, \cdots, k_{i-2}, k_{i-1}, k_i, k_{i+1}, k_{i+2}, \cdots, k_N) \\ I' &= (k_4, k_1, k_2, k_3, k_5, \cdots, k_{i-2}, k_{i+2}, k_{i-1}, k_i, k_{i+1}, \cdots, k_N) \end{aligned}$$

Figure 2. Shift operator operation

(3) Inversion Operator.

The inversion operation is a process where the genes in some substrings of a chromosome are inversed successively at a specific probability. The substrings for gene inversion in a chromosome and their length are selected randomly. The gene inversion can also be divided into single-point inversion and multi-point inversion. As to single-point inversion, only one substring of a chromosome is selected for gene inversion, and for multi-point inversion, multiple substrings of a chromosome are selected for gene inversion. The operation is shown in Figure 3. We assume O is a chromosome including multiple genes $l_i (i \in 1, 2, \cdots N)$. On the basis of the operations of single-point and multi-point inversions for chromosome O, we can obtain the chromosomes R and R'.

$$\begin{aligned} O &= (l_1, l_2, l_3, l_4, l_5, \cdots, l_{i-1}, l_i, l_{i+1}, \cdots, l_{j-1}, l_j, l_{j+1}, \cdots, l_N) \\ &\Downarrow \\ R &= (l_4, l_3, l_2, l_1, l_5, \cdots, l_{i-1}, l_i, l_{i+1}, \cdots, l_{j-1}, l_j, l_{j+1}, \cdots, l_N) \\ R' &= (l_4, l_3, l_2, l_1, l_5, \cdots, l_{i-1}, l_j, l_{j-1}, \cdots, l_{i+1}, l_i, l_{j+1}, \cdots, c_N) \end{aligned}$$

Figure 3. Inversion operator operation

The multi-point genetic operator is usually used when the chromosome length is large, while the single-point genetic operator is used when the chromosome length is small. Therefore, the genetic algorithms of single-point permutation, single-point shift, and single-point inversion for reproduction are used for genetic operator operations and applied to random selection in the general harmony search algorithm.

3.2.3. Flow of DEGA-HS Algorithm

As stated above, the mutation strategy of the differential mutation algorithm has the advantage of overcoming premature convergence and degeneration. Therefore, we considered introducing the mutation strategy of DE into the HS algorithm to overcome the premature convergence of HS, thus improving the convergence rate. At the same time, there is excessively high randomness in the generation of a new harmony in the general harmony search algorithm. High randomness brings both advantages and disadvantages. To balance this contradiction, the Partheno-genetic algorithm is also introduced.

The flow of the improved DEGA algorithm is as follows.

Step 1. Initialization of parameters.

In the new algorithm, the original tone pitch adjusting bandwidth (BW) will be canceled. Therefore, only the harmony memory size (HMS), harmony memory consideration rate (HMCR), pitch adjusting rate (PAR), and maximum times of creation (T_{max}) will be initialized.

Step 2. Random initialization of harmony memory.

Step 3. Generation of a new harmony.

For the general harmony search algorithm, there are mainly three operations to generate a new harmony at this step: memory consideration, pitch adjustment, and random selection. However, the operations of the proposed algorithm differ from them. The flow is as follows.

First, a random number $rand(1)$ is generated between (0,1). If $rand(1)$ is smaller than or equal to HMCR, the mutation strategy of "DE/best/1/bin" is used to generate a new harmony. Namely, $x_j^{new} = x_j^{best} + F(x_j^k - x_j^l)$. Then, a second random number $rand(2)$ is generated between (0,1). If $rand(2)$ is smaller than or equal to PAR, the mutation strategy of "DE/rand/1/bin" is used. Namely, $x_j^{new} = x_j^{rand} + F(x_j^k - x_j^l)$. Otherwise, no adjustment is made.

If $rand(1)$ is greater than HMCR, a new harmony is generated within the search boundary. Next, a third random number $rand(3)$ is generated between (0,1). If $rand(3)$ is smaller than or equal to PAR, a new harmony is generated according to the operation of the Partheno-genetic operation. Otherwise, no adjustment is made.

Step 4. Update of harmony memory.

The newly generated harmony is assessed. If it is better than the poorest harmony in the harmony memory, substitution is made.

Step 5. Judgment on termination.

Check whether current times of creation have reached the set maximum times of creation. If no, Steps 3–4 should be repeated until the maximum times of creation are reached.

3.3. Reward Population-Based Differential Genetic Harmony Search Algorithm (RDEGA-HS)

We hope the combination of the multi-population reward mechanism with a differential genetic harmony algorithm will increase the diversity of harmony and improve the convergence rate.

The specific flow of the algorithm is as follows.

Step 1. Initialization of algorithm parameters.

Step 2. Random initialization of harmony memory.

Step 3. The total harmony memory is divided into $Q + 1$ (Q is an integer greater than 0) sub-harmony memories, including Q common sub-harmony memories and one reward sub-memory.

Step 4. As to all sub-harmony memories including the reward sub-memory, new harmonies are generated on the basis of differential genetic operations.

Step 5. Update of each sub-harmony memory.

Step 6. Judge whether the maximum number T^1_{max} of iterations has been reached. If no, Steps 4 and 5 are repeated. Otherwise, move on to Step 7.

Step 7. Find the common sub-harmony memory with the best evolution effect at present, and combine it with the original reward harmony memory. Then, sort them from good to bad according to the fitness. On the premise of keeping the number of harmonies in the newly generated harmony memory unchanged, eliminate bad harmonies, and form a new reward harmony memory to replace the original reward harmony memory.

Step 8. Judge whether the upper limit T^2_{max} of iteration has been reached. If yes, then exit the loop and output the optimal solution. Otherwise, return to Step 4.

The pseudo-codes of the algorithm are shown in Algorithm 2.

Algorithm 2: RDEGA-HS Algorithm

Initialize the algorithm parameters
Initialize the total harmony memory
For $Pop = \cup Pop_i$
Repeat
  While time(2) $< T^2_{max}$, do
    Improvise each new harmony as follows:
    While time(1)$< T^1_{max}$
    For all j, do

$$x_j^{new} = \begin{cases} x_j^{best} + F(x_j^k - x_j^l) & \text{with probability HMCR} \\ \text{random selection} & \text{with probability 1-HMCR} \end{cases}$$

      If with probability HMCR, then

$$x'^{new}_j = \begin{cases} x_j^{rand} + F(x_j^k - x_j^l) & \text{with probability PAR} \\ x_j^{new} & \text{with probability 1-PAR} \end{cases}$$

      Else, do

$$x'^{new}_j = \begin{cases} operation\ of\ parthenogenetic & \text{with probability PAR} \\ x_j^{new} & \text{with probability 1-PAR} \end{cases}$$

      End If
    End For
    If the new harmony vector is better than the worst one in the
    original HM, then
      update the harmony memory
    End If
    End while
  Update the reward harmony memory
  End While
End For
Return best harmony

4. Experiments

To carry out an in-depth study on the performance of RDEGA-HS, two experiments were carried out to determine the influence of the values of HMS and HMCR on the performance of the algorithm and to perform a comparative study between the HS algorithm and the three proposed HS algorithm variants. The function name, formulation, and search range are listed in Table 1. In this paper, we used six benchmark functions to verify the ability of the RDEGA-HS algorithm in finding the optimal solution, so as to judge the advantages and disadvantages of the improved algorithm. Then, a comparison was made among the results of HS, RHS, and DEGA-HS.

Table 1. Standard test functions and parameter settings.

Function Name	Formulation	Search Range
Ackley	$f(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20$	$x_i \in [-32, 32]$
Griewank	$f(x) = 1 + \frac{1}{4000}\sum_{i=1}^{n}x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)$	$x_i \in [-600, 600]$
Rastrigin	$f(x) = \sum_{i=1}^{n}[x_i^2 - 10\cos(2\pi x_i) + 10]$	$x_i \in [-5.12, 5.12]$
Rosenbrock	$f(x) = \sum_{i=1}^{n}\left[100\left(x_{i+1} - x_i^2\right)^2 + (1 - x_i)^2\right]$	$x_i \in [-30, 30]$
Sphere	$f(x) = \sum_{i=1}^{n}x_i^2$	$x_i \in [-100, 100]$
Schwefel	$f(x) = \sum_{i=1}^{n}\lvert x\rvert + \prod_{i=1}^{n}\lvert x\rvert$	$x_i \in [-10, 10]$

The Ackley function is a widely used multi-mode test function. The Griewank function has many widely distributed local minimum values and is considered to be a convex function. The Rastrigin function is a high-dimensional multi-mode function; in this function, a frequent local minimum value is generated by increasing cosine modulation, and the locations of minimum values are generally scattered. In addition, benchmark functions such as Rosenbrock, Sphere, and Schwefel function are also used in the experimental steps.

4.1. Effects of HMS and HMCR on Performance of RDEGA-HS

In this section, the six benchmark functions in Table 1 are used for studying the influence of HMS and HMCR on the performance of the algorithm. The dimensions of the six functions were set to 40. Each case was independently run 50 times. Tables 2 and 3 report the average value and standard deviation obtained with different values of HMS and HMCR, respectively. In this experiment, we set the termination based on the maximum number of iterations. T^1_{max} is set to 10 and T^2_{max} is set to 100, so the maximum number of iterations is 1000.

It can be seen from Table 2 that HMS influenced the result of the algorithm. When HMS was 80, the algorithm achieved the best optimization accuracy and stability. Therefore, HMS was set to 80 in all test experiments herein.

Table 2. Effects of HMS on performance of RDEGA-HS (HMCR = 0.9).

Funtion	HMS					
	100	**80**	**50**	**20**	**10**	**5**
Ackley	9.26×10^{-3}	5.44×10^{-3}	1.24×10^{-2}	5.24×10^{-1}	1.02×10^{-1}	8.84
	1.20×10^{-2}	5.57×10^{-3}	8.45×10^{-2}	1.12×10^{-1}	9.54	2.84×10^{1}
Griewank	5.16×10^{-10}	1.35×10^{-11}	2.46×10^{-8}	9.54×10^{-6}	8.89×10^{-5}	5.12×10^{-2}
	9.25	1.68×10^{-1}	5.12	1.26×10^{1}	5.84×10^{1}	6.25×10^{1}
Rastrigin	1.91×10^{-7}	1.91×10^{-8}	1.91×10^{-6}	1.91×10^{-4}	1.91×10^{-2}	1.91
	3.21×10^{-3}	5.57×10^{-3}	1.05×10^{-3}	5.41×10^{-2}	4.89×10^{-1}	6.27
Rosenbrock	9.99×10^{-2}	8.68×10^{-2}	1.38×10^{-1}	7.54×10^{-1}	3.63	8.74
	7.45×10^{-1}	1.12×10^{-1}	4.21	2.84×10^{1}	9.79×10^{1}	1.84×10^{2}
Sphere	3.25×10^{-37}	1.03×10^{-40}	5.84×10^{-35}	6.84×10^{-34}	7.54×10^{-32}	8.47×10^{-30}
	6.45×10^{-29}	5.14×10^{-30}	7.48×10^{-29}	6.42×10^{-26}	3.41×10^{-24}	5.49×10^{-20}
Schwefel	5.84×10^{-34}	2.51×10^{-35}	4.98×10^{-32}	7.69×10^{-31}	3.98×10^{-30}	4.16×10^{-28}
	5.68×10^{-30}	4.87×10^{-31}	7.98×10^{-28}	1.36×10^{-27}	8.54×10^{-26}	2.65×10^{-24}

It can be seen from Table 3 that the value of HMCR obviously influenced the optimization result of the proposed algorithm. A greater HMCR will obviously improve the local

search ability, thus improving the performance of the algorithm. A smaller HMCR will influence the performance of the algorithm. When HMCR was 0.9, the proposed algorithm demonstrated a higher convergence rate. Therefore, in this paper, the HMCR value was chosen to be 0.9.

Table 3. Effects of HMCR on performance of RDEGA-HS (HMS = 80).

Function	HMCR					
	0.9	**0.8**	**0.6**	**0.5**	**0.3**	**0.1**
Ackley	4.26×10^{-3}	8.54×10^{-3}	9.98×10^{-3}	3.41×10^{-2}	9.88×10^{-2}	6.54
	9.25×10^{-4}	4.51×10^{-3}	5.89×10^{-2}	5.68×10^{-1}	1.25	6.66×10^{1}
Griewank	2.54×10^{-10}	7.84×10^{-8}	1.54×10^{-6}	5.65×10^{-5}	6.58×10^{-4}	2.58×10^{-3}
	7.33×10^{-2}	9.98×10^{-2}	1.54×10^{-1}	6.85	8.98	4.65×10^{1}
Rastrigin	9.91×10^{-9}	7.54×10^{-8}	8.54×10^{-7}	3.69×10^{-5}	9.58×10^{-4}	7.54×10^{-1}
	2.54×10^{-4}	8.96×10^{-4}	2.56×10^{-3}	6.66×10^{-3}	2.58×10^{-2}	8.95×10^{-2}
Rosenbrock	7.56×10^{-2}	8.75×10^{-2}	2.45×10^{-1}	9.87×10^{-1}	2.6	8.74×10^{1}
	1.58×10^{-1}	6.87×10^{-1}	8.95	6.54×10^{1}	1.26×10^{2}	9.68×10^{2}
Sphere	2.54×10^{-39}	8.54×10^{-38}	5.64×10^{-35}	9.87×10^{-34}	6.84×10^{-31}	5.36×10^{-29}
	7.84×10^{-30}	5.42×10^{-28}	6.46×10^{-28}	3.38×10^{-25}	5.10×10^{-23}	3.86×10^{-21}
Schwefel	5.98×10^{-36}	6.04×10^{-33}	9.40×10^{-31}	3.20×10^{-30}	1.65×10^{-28}	9.69×10^{-27}
	2.31×10^{-30}	3.94×10^{-29}	4.48×10^{-27}	9.45×10^{-25}	4.31×10^{-23}	1.41×10^{-22}

4.2. Comparative Study on RDEGA-HS and HS Variants

To study the expandability of the algorithm, six different variable dimensions were used in each function test: 30, 40, 50, 60, 70, and 80. In a simulation experiment, the common parameters were set as follows: HMS = 80, HMCR = 0.9, PAR = 0.6, and BW = 0.05. As in the previous experiment, we set the maximum number of iterations as the termination. Since the algorithm we proposed has two loops, in order to show fairness when compared with other algorithms, we set T^1_{max} to 10 and T^2_{max} to 100, so the maximum number of iterations is 1000. As a correspondence, the maximum number of iterations of the other three algorithms is also set to 1000. All optimization algorithms were achieved in Matlab R2021a.

To measure the convergence accuracy, robustness, convergence rate, and other performances of the different optimization algorithms, each function was run independently for 100 times. Then, the average optimal value and standard deviation were taken to serve as the final evaluation indexes. The curve of the average optimal fitness was given.

(1) Ackley Function

For the case where n = 40, the change in the average optimal fitness of the Ackley function with the number of iterations is shown in Figure 4.

(2) Griewank Function

For the case where n = 40, the change in the average optimal fitness value of the Griewank function with the number of iterations in shown in Figure 5.

(3) Rastrigin Function

For the case where n = 40, the change in the average optimal fitness value of the Rastrigin function with the number of iterations in shown in Figure 6.

(4) Rosenbrock Function

For the case where $n = 40$, the change in the average optimal fitness value of the Rosenbrock function with the number of iterations in shown in Figure 7.

(5) Sphere Function

For the case where n = 40, the change in the average optimal fitness value of the Sphere function with the number of iterations in shown in Figure 8.

(6) Schwefel Function

For the case where n = 40, the change in the average optimal fitness value of the Schwefel function with the number of iterations in shown in Figure 9.

It can be seen from the comparison of the optimization results of the test functions in Tables 4–9 and the changes in average fitness in Figures 4–9 that the proposed algorithm

has a strong optimization ability. Upon a general review of each algorithm's optimization process for the above six functions, the dimensions of the functions had a significant influence on the optimal value. As the dimensions increased, the optimal value of each algorithm also changed. For example, for the Ackley function, the variable dimensions had a significant influence on each algorithm. In terms of a low-dimensional function, RDEGA-HS and the other algorithms were all able to achieve the global optimum; however, in terms of a high-dimensional function, the RDEGA-HS algorithm still achieved a satisfactory result.

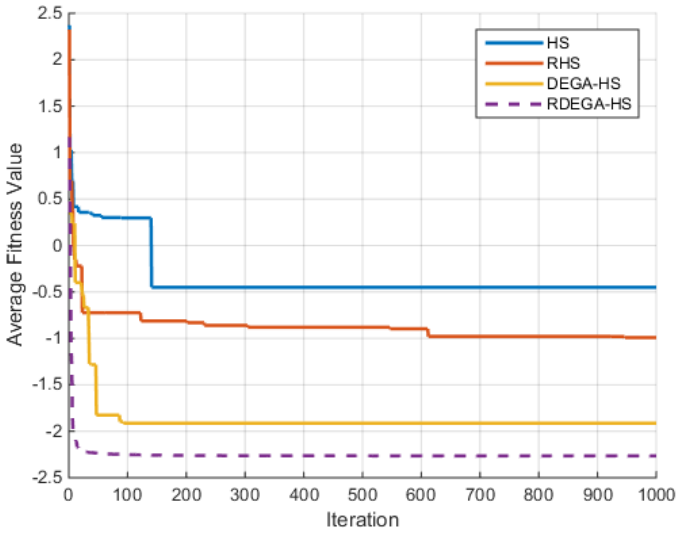

Figure 4. Change in average optimal fitness value of Ackley function with number of iterations.

Table 4. Result of the Ackley function under different optimization algorithms.

Algorithm	Dimensions	Theoretical Optimum	Mean Optimum	Standard Deviation
HS	30	0	3.66×10^{-2}	2.35
	40	0	3.55×10^{-1}	3.51
	50	0	1.25	4.21
	60	0	3.35	3.23×10^{1}
	70	0	8.70×10^{1}	5.64×10^{2}
	80	0	9.49×10^{2}	6.43×10^{2}
RHS	30	0	2.41×10^{-2}	5.33×10^{-2}
	40	0	1.02×10^{-1}	6.25×10^{-1}
	50	0	1.01	4.22
	60	0	8.25	1.10×10^{1}
	70	0	6.59×10^{1}	4.56×10^{2}
	80	0	1.88×10^{2}	3.12×10^{3}
DEGA-HS	30	0	8.20×10^{-5}	2.03×10^{-2}
	40	0	1.22×10^{-2}	6.01×10^{-2}
	50	0	1.25×10^{-1}	8.85×10^{-1}
	60	0	7.22	6.43
	70	0	6.96×10^{1}	7.80×10^{1}
	80	0	1.69×10^{2}	9.72×10^{3}
RDEGA-HS	30	0	2.67×10^{-5}	1.22×10^{-3}
	40	0	5.44×10^{-3}	3.25×10^{-3}
	50	0	9.56×10^{-2}	5.26×10^{-1}
	60	0	2.29×10^{-1}	4.14
	70	0	9.61×10^{-1}	2.34×10^{1}
	80	0	5.77	1.22×10^{2}

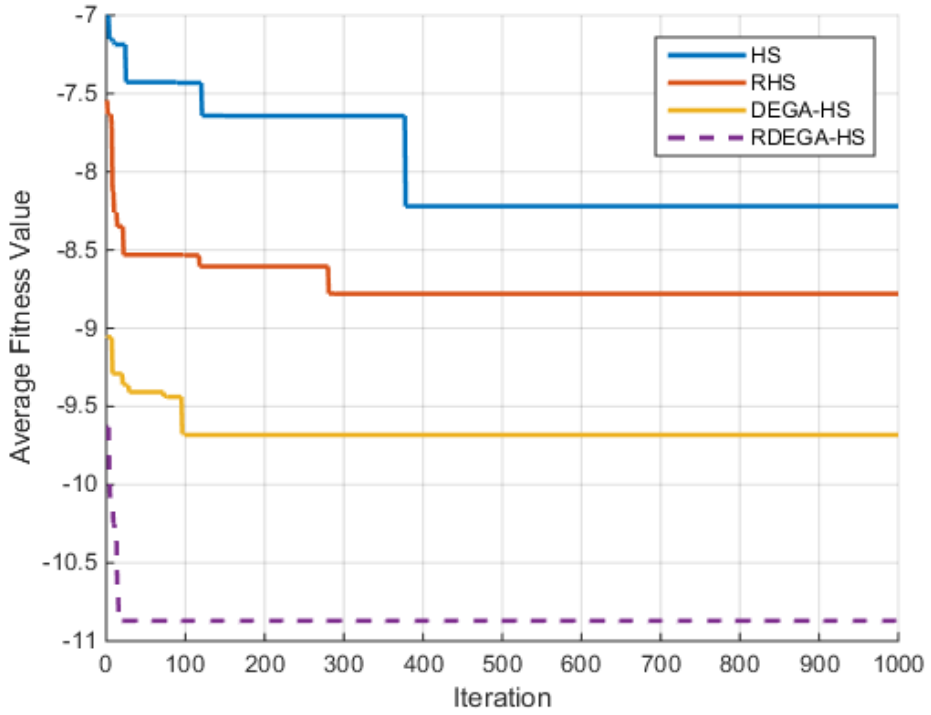

Figure 5. Change in average optimal fitness value of Griewank function with number of iterations.

Table 5. Results of the Griewank function under different optimization algorithms.

Algorithm	Dimensions	Theoretical Optimum	Average Optimum	Standard Deviation
HS	30	0	4.00×10^{-11}	1.00
	40	0	6.02×10^{-9}	1.50
	50	0	1.38×10^{-8}	2.64
	60	0	4.86×10^{-7}	7.16
	70	0	5.99×10^{-7}	5.99×10^{1}
	80	0	6.98×10^{-6}	7.94×10^{1}
RHS	30	0	3.13×10^{-10}	5.19×10^{-1}
	40	0	1.66×10^{-9}	7.73×10^{-1}
	50	0	7.84×10^{-9}	1.15
	60	0	2.06×10^{-8}	7.52
	70	0	6.34×10^{-8}	7.93
	80	0	4.54×10^{-7}	1.87×10^{1}
DEGA-HS	30	0	8.82×10^{-11}	2.15×10^{-1}
	40	0	2.08×10^{-10}	5.25×10^{-1}
	50	0	1.08×10^{-9}	7.96×10^{-1}
	60	0	8.90×10^{-8}	8.11×10^{-1}
	70	0	8.22×10^{-7}	4.48
	80	0	7.91×10^{-6}	5.21
RDEGA-HS	30	0	1.13×10^{-12}	1.24×10^{-2}
	40	0	1.35×10^{-11}	1.68×10^{-1}
	50	0	1.35×10^{-10}	2.29×10^{-1}
	60	0	7.44×10^{-9}	7.67×10^{-1}
	70	0	4.93×10^{-8}	9.45×10^{-1}
	80	0	8.01×10^{-7}	1.36

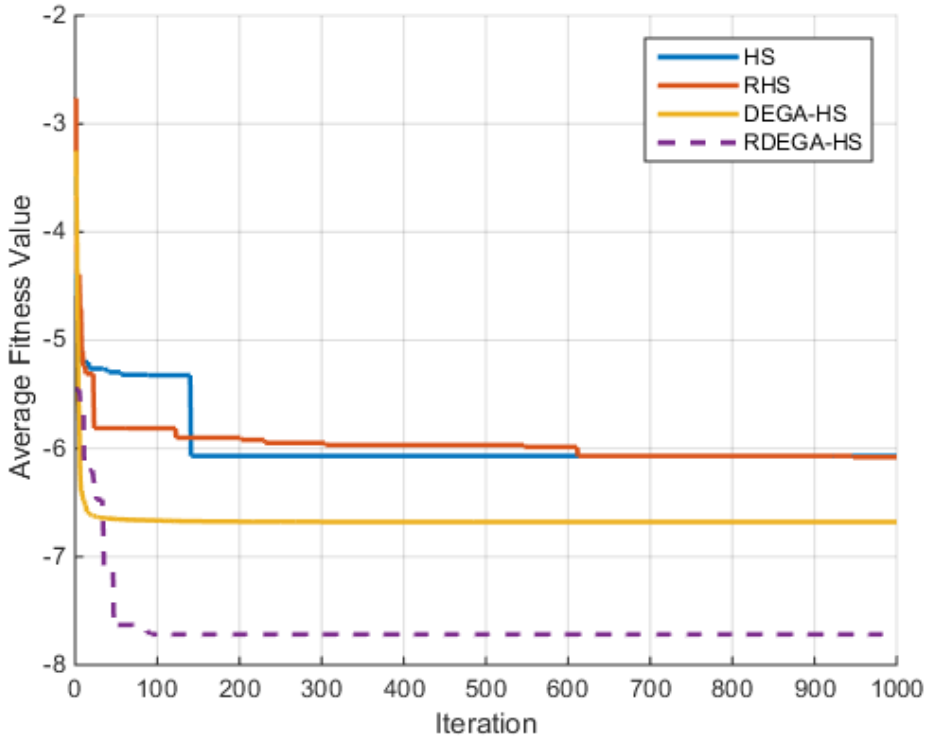

Figure 6. Change in average optimal fitness value of Rastrigin function with number of iterations.

Table 6. Results of the Rastrigin function under different optimization algorithms.

Algorithm	Dimensions	Theoretical Optimum	Average Optimum	Standard Deviation
HS	30	0	1.39×10^{-7}	3.32×10^{-2}
	40	0	8.52×10^{-7}	4.98
	50	0	2.86×10^{-5}	8.75
	60	0	5.20×10^{-5}	6.11×10^{1}
	70	0	5.38×10^{-4}	4.85×10^{2}
	80	0	3.59×10^{-3}	8.09×10^{2}
RHS	30	0	4.02×10^{-7}	1.72×10^{-1}
	40	0	8.30×10^{-7}	2.56
	50	0	3.12×10^{-5}	3.80
	60	0	4.32×10^{-5}	8.60
	70	0	2.73×10^{-4}	4.56×10^{1}
	80	0	5.91×10^{-3}	5.87×10^{2}
DEGA-HS	30	0	9.46×10^{-8}	7.12×10^{-2}
	40	0	2.09×10^{-7}	1.74×10^{-1}
	50	0	1.03×10^{-5}	2.64
	60	0	4.81×10^{-4}	7.34
	70	0	9.85×10^{-3}	1.67×10^{1}
	80	0	6.95×10^{-2}	5.74×10^{1}
RDEGA-HS	30	0	1.46×10^{-8}	4.12×10^{-3}
	40	0	1.91×10^{-8}	5.57×10^{-3}
	50	0	1.06×10^{-7}	7.60×10^{-1}
	60	0	8.34×10^{-7}	4.21
	70	0	4.63×10^{-6}	7.84
	80	0	9.85×10^{-6}	8.73

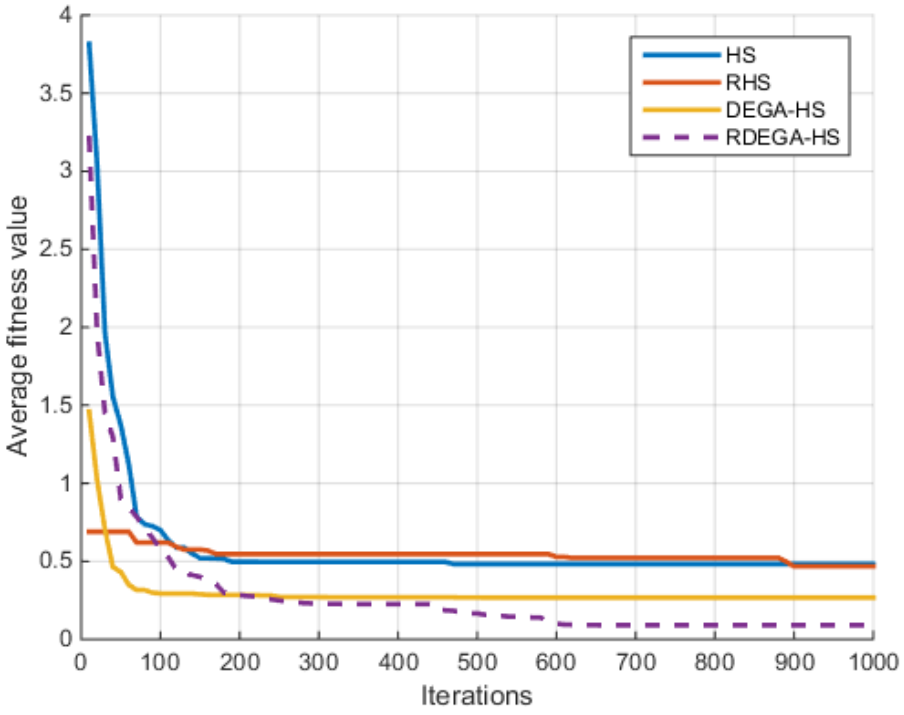

Figure 7. Change in average optimal fitness value of Rosenbrock function with number of iterations.

Table 7. Results of the Rosenbrock function under different optimization algorithms.

Algorithm	Dimensions	Theoretical Optimum	Average Optimum	Standard Deviation
HS	30	0	8.43×10^{-2}	6.36×10^{-2}
	40	0	4.82×10^{-1}	4.65×10^{-1}
	50	0	2.86	6.80
	60	0	7.14×10^{2}	9.45×10^{2}
	70	0	7.14×10^{2}	9.45×10^{2}
	80	0	5.23×10^{3}	7.35×10^{3}
RHS	30	0	4.08×10^{-2}	2.77×10^{-2}
	40	0	4.68×10^{-1}	8.47×10^{-1}
	50	0	6.93	3.72
	60	0	9.63	5.71×10^{1}
	70	0	3.72×10^{1}	6.62×10^{2}
	80	0	7.52×10^{2}	5.04×10^{3}
DEGA-HS	30	0	7.77×10^{-3}	7.98×10^{-1}
	40	0	2.67×10^{-1}	2.91
	50	0	6.12×10^{-1}	6.84
	60	0	1.55	8.15×10^{1}
	70	0	7.15×10^{1}	3.43×10^{2}
	80	0	7.02×10^{2}	1.48×10^{3}
RDEGA-HS	30	0	2.21×10^{-3}	9.98×10^{-2}
	40	0	9.04×10^{-2}	4.56×10^{-2}
	50	0	2.53×10^{-1}	9.10×10^{-1}
	60	0	5.90	5.78
	70	0	1.31×10^{1}	1.74×10^{1}
	80	0	8.33×10^{1}	4.77×10^{2}

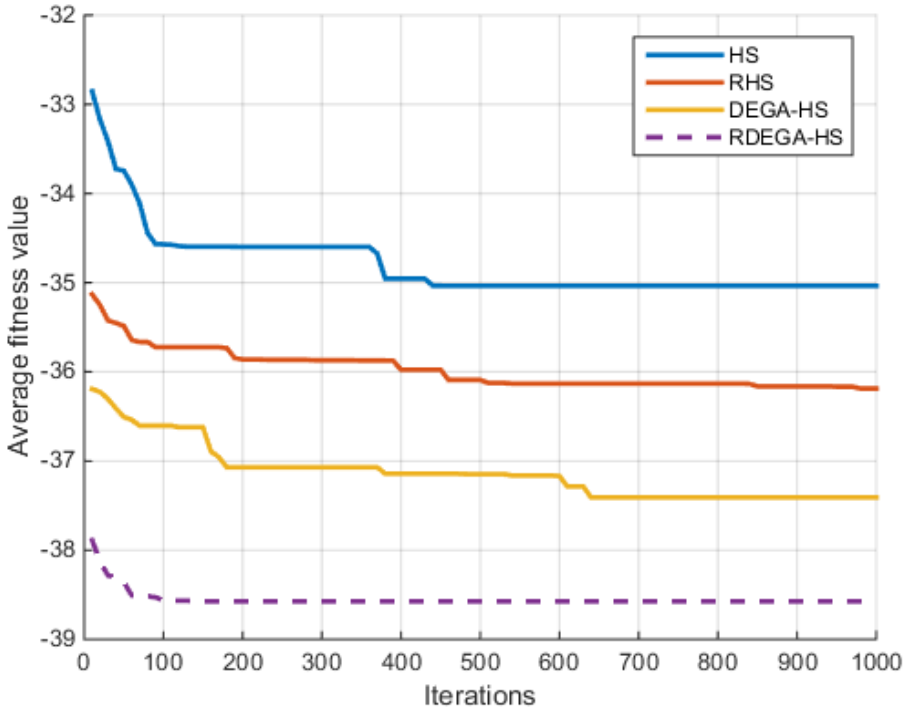

Figure 8. Change in average optimal fitness value of Sphere function with number of iterations.

Table 8. Results of the Sphere function under different optimization algorithms.

Algorithm	Dimensions	Theoretical Optimum	Average Optimum	Standard Deviation
HS	30	0	8.72×10^{-38}	3.26×10^{-28}
	40	0	9.21×10^{-36}	1.25×10^{-28}
	50	0	4.11×10^{-36}	3.67×10^{-28}
	60	0	8.42×10^{-34}	1.63×10^{-27}
	70	0	8.16×10^{-34}	1.09×10^{-26}
	80	0	2.91×10^{-33}	1.28×10^{-26}
RHS	30	0	2.45×10^{-38}	2.56×10^{-28}
	40	0	6.49×10^{-37}	2.06×10^{-28}
	50	0	1.14×10^{-35}	1.63×10^{-28}
	60	0	4.80×10^{-36}	3.32×10^{-27}
	70	0	7.41×10^{-35}	2.46×10^{-27}
	80	0	3.56×10^{-34}	2.33×10^{-27}
DEGA-HS	30	0	9.17×10^{-38}	3.63×10^{-29}
	40	0	3.90×10^{-38}	1.40×10^{-28}
	50	0	6.56×10^{-37}	3.06×10^{-28}
	60	0	8.62×10^{-35}	3.05×10^{-28}
	70	0	1.07×10^{-33}	1.73×10^{-27}
	80	0	1.14×10^{-32}	2.39×10^{-27}
RDEGA-HS	30	0	7.00×10^{-40}	6.58×10^{-31}
	40	0	2.66×10^{-39}	5.84×10^{-30}
	50	0	2.77×10^{-38}	2.27×10^{-29}
	60	0	3.92×10^{-36}	3.14×10^{-28}
	70	0	1.01×10^{-34}	3.69×10^{-28}
	80	0	3.89×10^{-34}	8.51×10^{-29}

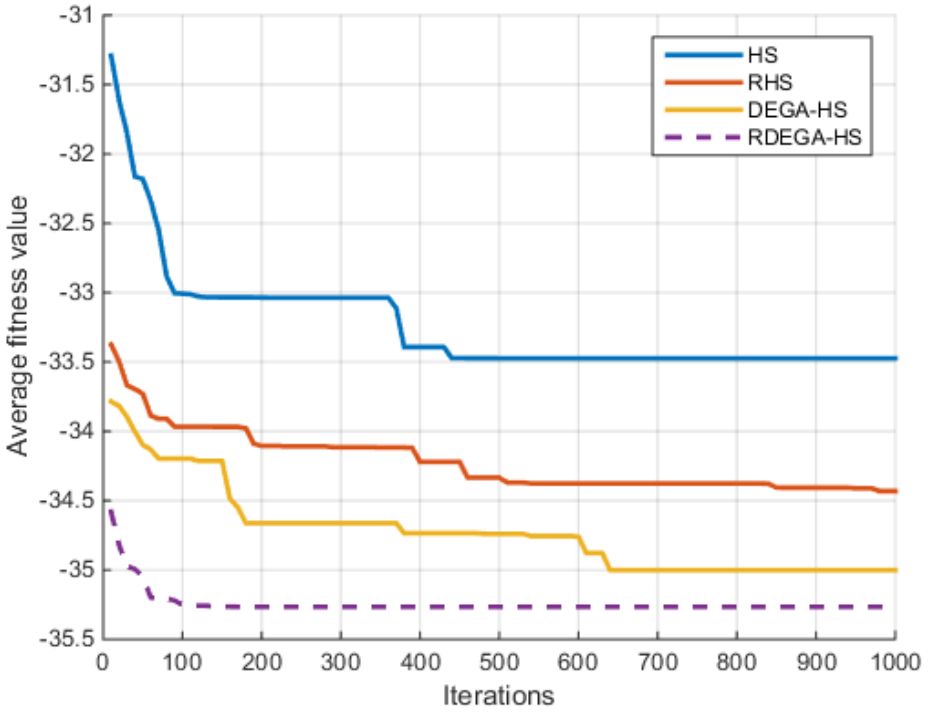

Figure 9. Change in average optimal fitness value of the Schwefel function with the number of iterations.

Table 9. Results of the Schwefel function under different optimization algorithms.

Algorithm	Dimensions	Theoretical Optimum	Average Optimum	Standard Deviation
HS	30	0	3.93×10^{-35}	7.20×10^{-27}
	40	0	3.36×10^{-34}	3.55×10^{-25}
	50	0	7.14×10^{-35}	3.82×10^{-26}
	60	0	2.59×10^{-33}	2.93×10^{-24}
	70	0	1.58×10^{-32}	3.07×10^{-23}
	80	0	5.24×10^{-31}	3.24×10^{-23}
RHS	30	0	2.44×10^{-36}	6.47×10^{-28}
	40	0	3.70×10^{-35}	1.69×10^{-26}
	50	0	1.60×10^{-34}	1.23×10^{-25}
	60	0	4.13×10^{-33}	3.01×10^{-25}
	70	0	2.17×10^{-32}	1.79×10^{-23}
	80	0	4.43×10^{-32}	3.38×10^{-23}
DEGA-HS	30	0	4.63×10^{-38}	9.70×10^{-29}
	40	0	9.97×10^{-36}	1.24×10^{-27}
	50	0	3.25×10^{-35}	8.49×10^{-27}
	60	0	1.13×10^{-33}	8.16×10^{-26}
	70	0	1.97×10^{-32}	3.24×10^{-24}
	80	0	5.93×10^{-32}	2.29×10^{-22}
RDEGA-HS	30	0	1.52×10^{-39}	2.19×10^{-29}
	40	0	5.42×10^{-36}	8.45×10^{-29}
	50	0	3.76×10^{-35}	3.19×10^{-26}
	60	0	6.41×10^{-35}	2.42×10^{-25}
	70	0	1.09×10^{-34}	1.53×10^{-25}
	80	0	3.20×10^{-33}	2.06×10^{-24}

As to a comparison of the algorithms' stability, the experiments were repeated 100 times. In addition to the optimization ability, which has received a lot of attention, the algorithms' stability is also an index worth paying attention to when evaluating the performance of

algorithms. After 100 repetitions, the variance in the experimental results was solved to check the fluctuation in the algorithms' optimal value. For the six functions, the RDEGA-HS algorithm still maintained satisfactory stability, while the rest of the algorithms were quite unstable.

Since five sub-harmony memories have been designed and calculated for RDEGA-HS, the running duration is longer than an algorithm where a single-harmony memory is set. However, as the diversity of harmony memories is increased, the local optimal solutions are avoided, thus improving the solving accuracy.

5. Conclusions

In this paper, on the basis of the standard harmony search (HS) algorithm, a reward population mechanism, differential mutation, and Partheno-genetic operation were used to improve the basic harmony search algorithm, whereby the harmony vector diversity and accuracy were improved. With the introduction of the reward population mechanism, the harmony memory was divided into four common sub-harmony memories and one reward harmony memory, each of which adopts the evolution strategy of the differential genetic harmony search. After each iteration, the sub-harmony memory with the best optimal average fitness was combined with the reward harmony memory. The mutation strategy of the differential evolution (DE) algorithm is able to overcome premature convergence. By introducing the mutation strategy of DE into the HS algorithm, it was able to overcome premature convergence and improve the convergence rate. Meanwhile, the Partheno-genetic algorithm was introduced the generation of new harmony vectors. Finally, the six frequently used test functions of Ackley, Griewank, Rastrigin, Rosenbrock, Sphere, and Schwefel were introduced to verify the validity and convergence of the algorithms. Then, a comparison was made on the calculation results of the standard harmony search algorithm, reward population-based harmony search algorithm, differential genetic harmony search algorithm, and reward population-based differential genetic harmony search algorithm. The result suggests that the reward population-based differential genetic harmony search algorithm has advantages such as a strong global search ability, high solving accuracy, and satisfactory stability.

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

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Snyman, J.A. *Practical Mathematical Optimization*; Springer: Boston, MA, USA, 2005; pp. 113–167.
2. Blum, C.; Puchinger, J.; Raidl, G.R.; Roli, A. Hybrid metaheuristics in combinatorial optimization: A survey. *Appl. Soft Comput.* **2011**, *11*, 4135–4151. [CrossRef]
3. Balamurugan, R.; Natarajan, A.M.; Premalatha, K. Stellar-mass black hole optimization for biclustering microarray gene expression data. *Appl. Artif. Intell.* **2015**, *29*, 353–381. [CrossRef]
4. Bianchi, L.; Dorigo, M.; Gambardella, L.M.; Gutjahr, W.J. A survey on metaheuristics for stochastic combinatorial optimization. *Nat. Comput.* **2009**, *8*, 239–287. [CrossRef]
5. Yang, X.S. *Nature-Inspired Metaheuristic Algorithms*; Luniver Press: Frome, UK, 2010; pp. 4–5.
6. Beheshti, Z.; Shamsuddin, S.M. A review of population-based meta-heuristic algorithms. *Int. J. Adv. Soft Comput. Appl.* **2013**, *5*, 1–35.
7. Yang, X.S.; Chien, S.F.; Ting, T.O. Bio-inspired computation and optimization: An overview. In *Bio-Inspired Computation in Telecommunications*; Yang, X.S., Chien, S.F., Ting, T.O., Eds.; Elsevier Inc.: New York, NY, USA, 2015; pp. 1–21.

8. Halim, A.H.; Ismail, I. Combinatorial optimization: Comparison of heuristic algorithms in travelling salesman problem. *Arch. Computat. Methods Eng*. **2019**, *26*, 367–380. [CrossRef]
9. Geem, Z.W.; Kim, J.H.; Loganathan, G.V. A new heuristic optimization algorithm: Harmony search. *Simulation* **2001**, *76*, 60–68. [CrossRef]
10. Zhao, F.Q.; Qin, S.; Yang, G.Q.; Ma, W.M.; Zhang, C.; Song, H.B. A differential-based harmony search algorithm with variable neighborhood search for job shop scheduling problem and its runtime analysis. *IEEE Access* **2018**, *6*, 76313–76330. [CrossRef]
11. Geem, Z.W. Optimal cost design of water distribution networks using harmony search. *Eng. Optimiz*. **2006**, *38*, 259–277. [CrossRef]
12. Geem Z.W. Harmony search applications in industry. In *Soft Computing Applications in Industry. Studies in Fuzziness and Soft Computing*; Prasad, B., Ed.; Springer: Berlin/Heidelberg, Germany, 2008; Volume 226, pp. 117–134.
13. Hasanipanah, M.; Keshtegar, B.; Thai, D.K; Troung, N.T. An ANN-adaptive dynamical harmony search algorithm to approximate the flyrock resulting from blasting. *Eng. Comput.* **2020**, 1–13 . [CrossRef]
14. Al-Omoush, A.A.; Alsewari, A.A.; Alamri, H.S.; Zamli, K.Z. Comprehensive review of the development of the harmony search algorithm and its applications. *IEEE Access* **2019**, *7*, 14233–14245. [CrossRef]
15. Mahdavi, M.; Fesanghary, M.; Damangir, E. An improved harmony search algorithm for solving optimization problems. *Appl. Math. Comput*. **2007**, *188*, 1567–1579. [CrossRef]
16. Khalili, M.; Kharrat, R.; Salahshoor, K.; Sefat, M.H. Global dynamic harmony search algorithm: GDHS. *Appl. Math. Comput*. **2014**, *228*, 195–219. [CrossRef]
17. Ouyang, H.B.; Gao, L.Q.; Kong, X.Y.; Guo, L. Application of MHS algorithm to structural design problems. *J. Northeast. Univ*. **2013**, *34*, 1687–1690. [CrossRef]
18. Pan, Q.K.; Suganthan, P.N.; Tasgetiren, M.F.; Liang, J.J. A self-adaptive global best harmony search algorithm for continuous optimization problems. *Appl. Math. Comput*. **2010**, *216*, 830–848. [CrossRef]
19. Zhang, K.L.; Chen, S.Y.; Shao, Z.Z. The improvement of harmony search algorithm. *Artif. Intell. Rob. Res*. **2015**, *4*, 32–39. [CrossRef]
20. Zhao, S.Z.; Liang, J.J.; Suganthan, P.N.; Tasgetiren, M.F. Dynamic multi-swarm particle swarm optimizer with harmony search. *Expert Syst. Appl*. **2011**, *38*, 3735–3742. [CrossRef]
21. Zhang, T.; Xu, X.Q.; Li, Z.H.; Abu-Siada, A.; Guo, Y.T. Optimum location and parameter setting of STATCOM based on improved differential evolution harmony search algorithm. *IEEE Access* **2020**, *8*, 87810–87819. [CrossRef]
22. Wu, X.L.; Li, J. Two layered approaches integrating harmony search with genetic algorithm for the integrated process planning and scheduling problem. *Comput. Ind. Eng*. **2021**, *155*, 107194. [CrossRef]
23. Moh'd Alia, O.; Mandava, R. The variants of the harmony search algorithm: An overview. *Artif. Intell. Rev*. **2011**, *36*, 49–68. [CrossRef]
24. Zhang, Y.; Li, J.C.; Li, L. An improved clustering-based harmony search algorithm (IC-HS). In Proceedings of the SAI Intelligent Systems Conference, Amsterdam, The Nederlands, 2–3 September 2021; Arai, K., Ed.; Springer: Cham, Germany, 2021; Volume 295, pp. 115–124.
25. Manjarres, D.; Landa-Torres, I.; Gil-Lopez, S.; Del Ser, J.; Bilbao, M.N.; Salcedo-Sanz, S.; Geem, Z.W. A survey on applications of the harmony search algorithm. *Eng. Appl. Artif. Intel*. **2013**, *26*, 1818–1831. [CrossRef]
26. Deng, W.; Shang, S.F.; Cai, X.; Zhao, H.M.; Song, Y.J.; Xu, J.J. An improved differential evolution algorithm and its application in optimization problem. *Soft Comput*. **2021**, *25*, 5277–5298. [CrossRef]
27. Jain, S.; Sharma, V.K.; Kumar, S. Robot path planning using differential evolution. In Proceedings of the Advances in Computing and Intelligent Systems, Beijing, China, 14–16 December 2019; Sharma, H., Govindan, K., Poonia, R., Kumar, S., El-Medany, W., Eds.; Springer: Singapore, 2020; pp. 531–537.
28. Li, M.J.; Tong, T.S. A partheno-genetic algorithm and analysis on its global convergence. *Acta Autom. Sin*. **1999**, *25*, 68–72. [CrossRef]
29. Yang, Z.; Li, J.C.; Li, L. Time-dependent theme park routing problem by Partheno-genetic algorithm. *Mathematics* **2020**, *8*, 2193. [CrossRef]