Hindawi Publishing Corporation
Discrete Dynamics in Nature and Society
Volume 2014, Article ID 573731, 12 pages
http://dx.doi.org/10.1155/2014/573731

Research Article

A Novel Discrete Global-Best Harmony Search Algorithm for Solving 0-1 Knapsack Problems

Wan-li Xiang, Mei-qing An, Yin-zhen Li, Rui-chun He, and Jing-fang Zhang

School of Traffic & Transportation, Lanzhou Jiaotong University, Lanzhou, Gansu 730070, China

Correspondence should be addressed to Wan-li Xiang; xiangwl@tju.edu.cn

Received 26 January 2014; Revised 19 March 2014; Accepted 25 March 2014; Published 24 April 2014

Academic Editor: Zbigniew Leśniak

In order to better solve discrete 0-1 knapsack problems, a novel global-best harmony search algorithm with binary coding, called DGHS, is proposed. First, an initialization based on a greedy mechanism is employed to improve the initial solution quality in DGHS. Next, we present a novel improvisation process based on intuitive cognition of improvising a new harmony, in which the best harmony of harmony memory (HM) is used to guide the searching direction of evolution during the process of memory consideration, or else a harmony is randomly chosen from HM and then a discrete genetic mutation is done with some probability during the phase of pitch adjustment. Third, a two-phase repair operator is employed to repair an infeasible harmony vector and to further improve a feasible solution. Last, a new selection scheme is applied to decide whether or not a new randomly generated harmony is included into the HM. The proposed DGHS is evaluated on twenty knapsack problems with different scales and compared with other three metaheuristics from the literature. The experimental results indicate that DGHS is efficient, effective, and robust for solving difficult 0-1 knapsack problems.

1. Introduction

Since the last four decades, the zero-one knapsack problem, inverse $\{0, 1\}$-knapsack problem, and their variants have attracted much attention [1–24]. This is the reason why they play an important role in computing theory and in a number of real-world applications, such as project selection, resource allocation, production planning, and others [24, 25].

In this paper, we focus on studying the classical 0-1 knapsack problem, where a set of n items is given and each item j has a profit p_j and a weight w_j. And the problem is to choose a subset of the items such that the profit sum of the chosen items is maximized without exceeding the capacity C. Mathematically, the classical zero-one knapsack problem can be modelled as the following integer linear programming model:

$$\begin{aligned} \max \quad & f(\vec{x}) = \sum_{j=1}^{N} \left(p_j x_j\right) \\ \text{s.t.} \quad & \sum_{j=1}^{N} \left(w_j x_j\right) \leqslant C \\ & x_j \in \{0, 1\}, \quad j = 1, 2, \ldots, N, \end{aligned} \tag{1}$$

where x_j takes the value one if and only if the item j is loaded, coefficients p_j and w_j represent a profit and weight of item j, respectively, C is a constant denoting the capacity of the corresponding knapsack, and N is the number of items. Without loss of generality, it is assumed that all coefficients p_j, w_j, and C are positive. Meantime, we suppose that $w_j \leqslant C$, $\forall j \in [1 \cdots N]$ and that $\sum_{j=1}^{N} w_j > C$.

As far as 0-1 knapsack problem is concerned, there are essentially two types of algorithms: exact algorithms and heuristic algorithms. The exact approaches for knapsack problems mainly include dynamic programming, branch and bound, and its enhanced variant: branch and cut [4–7, 9].

Unfortunately, the 0-1 knapsack problem belongs to the class of $\mathcal{NP}$-hard problems [12]. That is to say, the number of alternate optimal solutions for the problem grows exponentially with the problem sizes. Hence, $\mathcal{NP}$-hard problems are very difficult to solve and there is no polynomial time algorithm for solving them. Thus, more and more researchers start to investigate the 0-1 knapsack problems by means of heuristics, metaheuristics, and hybridizing algorithms based on exact algorithms and heuristic algorithms [11–15, 20, 22, 24]. For example, He et al. [11] proposed a greedy genetic

```
(1) for j = 1 to D do
(2)    //memory consideration
(3)    if rand(0, 1) ⩽ HMCR then
(4)       Choose a harmony x_i from HM randomly, i ∈ [1..HMS]
(5)       x'_j = x_{i,j} // a new harmony x' = (x'_1, ..., x'_D)
          // pitch adjustment
(6)       if rand(0, 1) ⩽ PAR then
(7)          x'_j = x'_j ± rand (0, 1) · bw // bw is a constant to be predetermined
(8)       end if
(9)    else
(10)      x'_j = L_j + rand(0, 1) · (U_j − L_j) // random selection
(11)   end if
(12) end for
```

Algorithm 1: The procedure of improvising a new harmony.

algorithm (GGA) for 0-1 knapsack problems. In GGA, a novel greedy operator and a repair operator are introduced to speed up the performance of standard genetic algorithm. Bansal and Deep [12] proposed a modified binary particle swarm optimization (MBPSO) for solving 0-1 knapsack problems and multidimensional knapsack problems. When compared with basic binary particle swarm optimization (BPSO), MBPSO has achieved better performance. Inspired by the nature of chemical reaction, Truong et al. [13] proposed a chemical reaction optimization with greedy strategy algorithm (CROG) to solve the zero-one knapsack problems. However, it has seven parameters to be predefined and the parameter settings are not easy. In addition, a novel global harmony search algorithm, named NGHS, was first proposed by Zou et al. [24] to solve knapsack problems. In NGHS, an adaptive step scheme for jth decision variable, a genetic mutation operation, and a discrete technique for real coded NGHS are introduced. Later, Layeb [14] presented a hybrid algorithm based on harmony search (HS) algorithm and quantum computing, which is called quantum inspired harmony search algorithm (QIHSA) for solving the knapsack problems. Meanwhile, Wang et al. [15] proposed an improved adaptive binary harmony search (ABHS) algorithm for solving binary knapsack problems. Although the recent proposed algorithms have been improved to some extent, their convergence speed, convergence precision, and robustness are waiting to be further enhanced.

Harmony search (HS) algorithm, a novel population-based evolutionary algorithm, was first proposed by Geem et al. [26] in 2001. Due to its simplicity and ease of implementation, it has aroused great interest and has been successfully applied to solve a variety of optimization problems including many real parameter optimization problems [27–30] and knapsack problems [15, 24, 31–35] during the last decades.

However, the binary coded harmony search algorithm has only begun as stated in [15]. In other words, the convergence performance of HS and its variants is necessary to be further enhanced. Therefore, in order to further improve the convergence performance of HS, a novel discrete global-best harmony search algorithm, called DGHS, is proposed for 0-1 knapsack problems. In DGHS, an initialization based on a greedy operation, a novel improvisation process for generating a new harmony, and a two-phase repair operator are integrated. Then, experimental results tested on twelve benchmark instances with small or medium sizes and eight randomly generated instances with high dimensions show that DGHS is superior to both GHS and NGHS in most cases.

The rest of the paper is organized as follows. In Section 2, the standard harmony search algorithm is described briefly. An overview of the global-best harmony search (GHS) algorithm proposed by Omran and Mahdavi [35] is given in Section 3. Section 4 describes the proposed DGHS in detail. In Section 5, extensive computational experiments are presented and discussed. Finally, some conclusions are drawn in the Section 6.

2. Harmony Search Algorithm

Harmony search (HS) algorithm was developed by Geem et al. [26] in 2001 through mimicking the improvisation process of music players. Like many other population based metaheuristics, it works as an iterative process for an optimization problem. And the optimization problem can be formulated as follows:

$$\min\ f(\vec{x}), \tag{2}$$

where $\vec{x} = (x_1, x_2, \ldots, x_D) \in \mathfrak{R}^D$ and the feasible solution space can be denoted by $\Omega = \prod_{j=1}^{D}[L_j, U_j]$. Moreover, L_j and U_j represent the lower bound and upper bound of the jth decision variable, respectively.

In general, its procedure consists of the following four steps.

Step 1. Initialize the control parameters and a harmony memory. At the step, an initial harmony memory (HM) is filled with a population of HMS (harmony memory size) harmonies generated randomly. In addition, the parameters of HS, that is, harmony memory consideration rate HMCR and pitch adjusting rate PAR, are given in advance.

Step 2. Improvise a new harmony from the current HM. And the details of the procedure can be given in Algorithm 1.

```
(1) for j = 1 to D do
(2)    // memory consideration
(3)    if rand(0, 1) ⩽ HMCR then
(4)       Choose a harmony x_i from HM randomly, i ∈ [1..HMS]
(5)       x'_j = x_{i,j}
          // pitch adjustment
(6)       if rand(0, 1) ⩽ PAR(t) then
(7)          Generate a random integer number k ∈ [1..D]
             // best represents the index of the best harmony in the HM
(8)          x'_j = x_{best,k}
(9)       end if
(10)   else
(11)      x'_j = L_j + rand(0, 1) · (U_j − L_j) // random selection
(12)   end if
(13) end for
```

Algorithm 2: The procedure of improvising a new harmony of GHS.

Step 3. If the new generated harmony is better than the worst one in HM, then replace the worst harmony with the new one; otherwise, go to the next step.

Step 4. If a stopping criterion is not satisfied, go to Step 2.

In addition, more detailed information about the original HS can be found in [26].

3. Global-Best Harmony Search Algorithm

To further improve the convergence performance of HS and overcome some shortcomings of HS, a new variant of HS, called GHS, was proposed by Omran and Mahdavi [35].

First, the GHS dynamically updates parameter PAR according to the following equation:

$$\mathrm{PAR}(t) = \mathrm{PAR}_{\min} + \frac{\mathrm{PAR}_{\max} - \mathrm{PAR}_{\min}}{\mathrm{NI}} t, \tag{3}$$

where PAR(t) represents the pitch adjusting rate at generation t, $\mathrm{PAR}_{\min}$ and $\mathrm{PAR}_{\max}$ are the minimum and maximum adjusting rate, respectively, t is the iterative variable, and NI is the number of improvisations.

Second, GHS modifies the pitch adjustment step of HS in order to take advantage of the guiding information of the best harmony in the HM. Furthermore, GHS excludes the parameter bw at the phase.

Last, from the above mentioned explanation, it can be concluded that GHS has the same steps as HS with the exception that the process of improvising a new harmony is modified as shown in Algorithm 2.

Due to the guiding information of the best harmony x_{best} in HM, GHS outperforms both HS and IHS [36]. What is more, Omran and Mahdavi employed GHS to solve integer programming problems. The GHS developed for real search spaces can be utilized to solve integer programming problems by rounding off the real optimal values to the nearest integers. It should be noted that a penalty function method is also used here to make GHS solve the knapsack problems. Clearly, the solution search space for 0-1 knapsack problems should be $\Omega = [0, 1]^N$.

4. A Novel Discrete Global-Best Harmony Search Algorithm

The original HS is good at identifying the high performance regions of the solution space in a reasonable time but poor at performing local search for numerical optimization problems [37]. Namely, there is imbalance between the exploration and the exploitation of HS. Furthermore, HS designed for continuous space cannot be directly used to solve discrete combinatorial optimization problems.

In order to overcome the drawbacks of HS, a novel discrete global-best harmony search (DGHS for short) algorithm is particularly designed for binary optimization problems in this paper.

Owing to better performance of GHS, some modifications to GHS are introduced to further enhance the convergence performance of GHS. Then a novel binary coded GHS, a two-phase repair operator, and a new greedy selection mechanism are integrated into the DGHS. And they are described in detail as follows.

4.1. Initialization in DGHS. The initial population in DGHS is generated randomly using a Bernoulli process. Specifically, for each decision variable of an initial harmony vector, a number within [0, 1] is generated randomly. If the value of the number is less than 0.5, the corresponding variable in DGHS takes 0; otherwise it takes 1. In this way, a set of HMS harmonies will be generated randomly.

In addition, another harmony vector $\vec{x}_o$ is generated based on a greedy operation. The greedy operation is based on the idea that the item with higher profit density ratio should be packed first. And the profit density ratio μ_j can be calculated by the following equation:

$$\mu_j = \frac{p_j}{w_j}. \tag{4}$$

(1) Record the best harmony in the HM, and its index is represented by *best*
(2) **for** $j = 1$ to D **do**
(3) // memory consideration
(4) **if** rand$(0, 1) \leqslant$ HMCR(t) **then**
(5) $x'_j = x_{\text{best},j}$
(6) **else**
(7) Generate a random integer number $a \in [1..\text{HMS}] \wedge a \neq best$
(8) $x'_j = x_{a,j}$
// pitch adjustment for the pitch chosen randomly
(9) **if** rand $(0, 1) \leqslant$ PAR **then**
(10) $x'_j = \left|x'_j - 1\right|$ // discrete genetic mutation
(11) **end if**
(12) **end if**
(13) **end for**

Algorithm 3: The procedure of improvising a new harmony of DGHS.

The first way is to sort the items by u_j. Then we add the items with higher value of u_j until the total weight exceeds the capacity of the knapsack. Thus, we can get a harmony vector $\vec{x}_o$. And if $\vec{x}_o$ is better than the worst one of previously initialized HM, then substitute the worst one with $\vec{x}_o$.

4.2. Dynamically Updating of the Parameters. First, to our knowledge, the control parameters HMCR and PAR play an important role in standard HS. More specifically, the parameters HMCR and PAR set to constants may have some adverse effects on the performance of HS, which is the idea behind designing the varying parameters. For HS with the guidance of the best harmony, the parameter HMCR with a larger value can be helpful to accelerate the convergence speed of HS variants with the guidance of the best harmony (individual) at the beginning of search, while the parameter HMCR with a smaller value can help the corresponding HS variant to get out of local minima at the end of search. Like the parameter HMCR, a varying parameter PAR is also considered in this paper in order to further balance the exploration and exploitation of HS variants.

Second, two dynamically updating schemes of the parameters HMCR and PAR are designed in order to improve the performance of GHS. And the two schemes can be described as follows:

$$\text{HMCR}(t) = \text{HMCR}_{\max} - \frac{\text{HMCR}_{\max} - \text{HMCR}_{\min}}{\text{NI}} t, \quad (5a)$$

$$\text{PAR}(t) = \text{PAR}_{\max} - \frac{\text{PAR}_{\max} - \text{PAR}_{\min}}{\text{NI}} t, \quad (5b)$$

where $\text{HMCR}_{\max}$ and $\text{HMCR}_{\min}$ represent the lower and upper bounds of HMCR, respectively. And the other parameters are the same as those in (3).

Last but not least, we found that the parameter PAR with a constant value can work better in DGHS through a lot of experiments. It should be noticed that the parameter PAR is set to a constant, that is, 0.75, in the later study.

4.3. A Novel Scheme of Improvising a New Harmony. At first, musicians most likely choose a perfect state of a harmony from their memory or harmony memory during the process of improvising a new harmony. Next, they may select a pitch from the current harmony memory randomly and then they would perform a fine tune operation, that is, pitch adjustment, for the chosen pitch to improve the effectiveness of music. In addition, as far as knapsack problem is concerned, the states of a pitch just include zero and 1; that is, any state of a pitch is ranging in $\{0, 1\}$. So discrete genetic mutation used in [24] is suitable for pitch adjustment. Based on the intuitional idea and the aforementioned explanation, a novel scheme of improvising a new harmony can be given in Algorithm 3.

4.4. Two-Phase Repair Operator. A major drawback of penalty function is that it needs to preset a very large constant, which is yet problem-dependent. In order to reduce the penalty coefficient, a repair operator is introduced. For DGHS, new generated harmony vector needs to be repaired under two cases. One is that the harmony vector violates the constraints. The other is that the knapsack corresponding to the new generated harmony vector can still pack other items without exceeding the capacity of knapsack [11, 13]. Hence, the repair operator consists of two phases. The first phase, called `DROP`, is responsible for repairing a harmony vector violating the constraint. The second phase, named `ADD`, mainly takes charge of optimizing a new generated harmony vector whose total weight is less than the capacity of knapsack. It is worth mentioning that the "`DROP`" phase should be performed first, and then the "`ADD`" phase is carried out. This is because a harmony vector changed after previously performing "`DROP`" phase becomes feasible, but its total weight may be less than the capacity C. Thus, The "`ADD`" phase has to be performed on the harmony vector again to improve the solution quality. The detailed pseudocode for the repair operator is shown in Algorithm 4.

4.5. A New Selection Mechanism. In order to avoid being clustered in the best harmony, we employ a new selection mechanism, in which a new generated harmony vector $\vec{x}'$ is compared with $\vec{x}_{\text{best}}$ first, and if $\vec{x}'$ is better than $\vec{x}_{\text{best}}$, replace $\vec{x}_{\text{best}}$ with $\vec{x}'$; otherwise, $\vec{x}'$ is compared with the worst

(1) Let v represent a new generated harmony vector
// Calculate the total weight of knapsack according to v
(2) $totW = v * w^T$ // $totW$ denotes the total weight
(3) **if** $totW > C$ **then**
(4) // The "DROP" phase
(5) **for** $j = 1$ to N **do**
(6) $\lambda_j = \frac{p_j}{w_j} v_j$ // Compute the profit density value of items loaded
(7) **end for**
(8) Sort items in increasing order of λ_j, and let $\widetilde{\lambda}_j$ represent the result sorted, and ind_j denotes the original index of each $\widetilde{\lambda}_j$
(9) **for** $j = 1$ to N **do**
(10) **if** $\widetilde{\lambda}_j == 0$ **then**
(11) Continue
(12) **end if**
(13) $v_{\text{ind}_j} = 0$ // Unload the ind_jth item from the knapsack
(14) $totW = v * w^T$
(15) **if** $totW \leqslant C$ **then**
(16) Break // Terminate the "DROP" phase of repair process
(17) **end if**
(18) **end for**
(19) **end if**
(20) $totW = v * w^T$
(21) **if** $totW < C$ **then**
(22) // The "ADD" phase
(23) Calculate the profit density ratio μ_j according to (4)
(24) Sort all the items in decreasing order of μ_j, and let $\widetilde{\mu}_j$ represent the result sorted, and ind_j denotes the original index of each $\widetilde{\mu}_j$
(25) **for** $j = 1$ to N **do**
(26) **if** $v_{\text{ind}_j} == 0$ **then**
(27) Let v^1 represent a temporary harmony vector, and set $v^1 = v$
(28) Set $v^1_{\text{ind}_j} = 1$ //Try to load the ind_jth item
(29) **if** $v^1 * w^T \leq C$ **then**
(30) $v_{\text{ind}_j} = 1$ // Load the ind_jth item into knapsack
(31) **end if**
(32) **end if**
(33) **end for**
(34) **end if**

Algorithm 4: The two-phase repair operator.

harmony $\vec{x}_{\text{worst}}$ in HM again, and a greedy selection is applied between $\{\vec{x}', \vec{x}_{\text{worst}}\}$. In this way, the number of harmony around the best harmony $\vec{x}_{\text{best}}$ would be small at the early stage of evolution so that the diversity of harmony memory would be kept better. As a consequence, DGHS not only speeds up the convergence speed but also avoids being trapped in a local optimum.

4.6. The Proposed Algorithm. According to the analysis and modifications mentioned above, an initialization of HM based on greedy operation, a novel scheme of improvising a new harmony with the direction information of the best harmony, and a repair operator with greedy strategy make up the proposed DGHS designed for binary knapsack problems. The pseudocode of DGHS is given in Algorithm 5.

5. Experimental Results and Analysis

5.1. Benchmark Instances and Parameter Settings. In order to evaluate the performance of DGHS, twelve benchmark instances chosen from [11, 13–15, 24] are employed here to validate its performance. The detailed information of the twelve test problems is listed in Tables 1 and 2. In addition, eight randomly generated test instances with large scales are also employed here to further verify the validity of DGHS according to a rule used in [24]. Concretely, for each N (the dimensions or the number of items), the weight w_j ($j = 1, 2, \ldots, N$) is randomly generated in the range of $[5, 20]$, and the profit p_j ($j = 1, 2, \ldots, N$) is randomly produced between $[50, 100]$, and the detailed settings of capacity of knapsack C are given in Table 3. In short, twenty test instances are employed to testify the performance of DGHS thoroughly.

(1) Set the harmony memory size HMS, the number of maximum improvisations maxFEs, and other control parameters
(2) Initialize the harmony memory HM, and perform Algorithm 4 to repair the harmony vector of HM, then evaluate their objective function values
(3) Set FEs = 1 // FEs represents the iterative variable
(4) **while** FEs $\leqslant$ maxFEs **do**
(5) Record the position of the best harmony in the HM, and its index is represented by *best*, likewise, *worst* denotes the index of the worst harmony in the current HM
(6) Calculate the parameter HMCR(FEs) according to (5a)
(7) Perform Algorithm 3 to produce a new harmony vector $\vec{x}'$
(8) Perform Algorithm 4 to repair the new harmony vector $\vec{x}'$
// Perform a new greedy selection scheme
(9) **if** $\vec{x}'$ is better than or equal to $\vec{x}_{best}$ **then**
(10) Replace $\vec{x}_{best}$ with $\vec{x}'$
(11) **else if** $\vec{x}'$ is better than or equal to $\vec{x}_{worst}$ **then**
(12) Substitute $\vec{x}_{worst}$ with $\vec{x}'$
(13) **end if**
(14) Memorize the best harmony achieved so far
(15) Set FEs = FEs + 1
(16) **end while**

Algorithm 5: The DGHS algorithm.

Table 1: The dimensions and parameters of the twelve test problems.

No.	Dim.	maxFEs	Parameter
Kp1	10	1000	w = (95, 4, 60, 32, 23, 72, 80, 62, 65, 46); C = 269; p = (55, 10, 47, 5, 4, 50, 8, 61, 85, 87)
Kp2	20	1000	w = (92, 4, 43, 83, 84, 68, 92, 82, 6, 44, 32, 18, 56, 83, 25, 96, 70, 48, 14, 58); C = 878; p = (44, 46, 90, 72, 91, 40, 75, 35, 8, 54, 78, 40, 77, 15, 61, 17, 75, 29, 75, 63)
Kp3	4	1000	w = (6, 5, 9, 7); C = 20; p = (9, 11, 13, 15)
Kp4	4	1000	w = (2, 4, 6, 7); C = 11; p = (6, 10, 12, 13)
Kp5	15	1000	w = (56.358531, 80.874050, 47.987304, 89.596240, 74.660482, 85.894345, 51.353496, 1.498459, 36.445204, 16.589862, 44.569231, 0.466933, 37.788018, 57.118442, 60.716575); C = 375; p = (0.125126, 19.330424, 58.500931, 35.029145, 82.284005, 17.410810, 71.050142, 30.399487, 9.140294, 14.731285, 98.852504, 11.908322, 0.891140, 53.166295, 60.176397)
Kp6	10	1000	w = (30, 25, 20, 18, 17, 11, 5, 2, 1, 1); C = 60; p = (20, 18, 17, 15, 15, 10, 5, 3, 1, 1)
Kp7	7	1000	w = (31, 10, 20, 19, 4, 3, 6); C = 50; p = (70, 20, 39, 37, 7, 5, 10)
Kp8	23	1000	w = (983, 982, 981, 980, 979, 978, 488, 976, 972, 486, 486, 972, 972, 485, 485, 969, 966, 483, 964, 963, 961, 958, 959); C = 10000; p = (981, 980, 979, 978, 977, 976, 487, 974, 970, 485, 485, 970, 970, 484, 484, 976, 974, 482, 962, 961, 959, 958, 857)
Kp9	5	1000	w = (15, 20, 17, 8, 31); C = 80; p = (33, 24, 36, 37, 12)
Kp10	20	1000	w = (84, 83, 43, 4, 44, 6, 82, 92, 25, 83, 56, 18, 58, 14, 48, 70, 96, 32, 68, 92); C = 879; p = (91, 72, 90, 46, 55, 8, 35, 75, 61, 15, 77, 40, 63, 75, 29, 75, 17, 78, 40, 44)
Kp11	50	1000	w = (80, 82, 85, 70, 72, 70, 66, 50, 55, 25, 50, 55, 40, 48, 50, 32, 22, 60, 30, 32, 40, 38, 35, 32, 25, 28, 30, 22, 25, 30, 45, 30, 60, 50, 20, 65, 20, 25, 30, 10, 20, 25, 15, 10, 10, 10, 4, 4, 2, 1); C = 1000; p = (220, 208, 198, 192, 180, 180, 165, 162, 160, 158, 155, 130, 125, 122, 120, 118, 115, 110, 105, 101, 100, 100, 98, 96, 95, 90, 88, 82, 80, 77, 75, 73, 72, 70, 69, 66, 65, 63, 60, 58, 56, 50, 30, 20, 15, 10, 8, 5, 3, 1)
Kp12	100	1000	w = (54, 183, 106, 82, 30, 58, 71, 166, 117, 190, 90, 191, 205, 128, 110, 89, 63, 6, 140, 86, 30, 91, 156, 31, 70, 199, 142, 98, 178, 16, 140, 31, 24, 197, 101, 73, 169, 73, 92, 159, 71, 102, 144, 151, 27, 131, 209, 164, 177, 177, 129, 146, 17, 53, 164, 146, 43, 170, 180, 171, 130, 183, 5, 113, 207, 57, 13, 163, 20, 63, 12, 24, 9, 42, 6, 109, 170, 108, 46, 69, 43, 175, 81, 5, 34 146, 148, 114, 160, 174, 156, 82, 47, 126, 102, 83, 58, 34, 21, 14); C = 6718; p = (597, 596, 593, 586, 581, 568, 567, 560, 549, 548, 547, 529, 529, 527, 520, 491, 482, 478, 475, 475, 466, 462, 459, 458, 454, 451, 449, 443, 442, 421, 410, 409, 395, 394, 390, 377, 375, 366, 361, 347, 334, 322, 315, 313, 311, 309, 296, 295, 294, 289, 285, 279, 277, 276, 272, 248, 246, 245, 238, 237, 232, 231, 230, 225, 192, 184, 183, 176, 174, 171, 169, 165, 165, 154, 153, 150, 149, 147, 143, 140, 138, 134, 132, 127, 124, 123, 114, 111, 104, 89, 74, 63, 62, 58, 55, 48, 27, 22, 12, 6)

Table 2: The detailed information of the optimal solutions obtained so far on the twelve test problems.

No.	Optimal solution	Optimal Value	Value of constraint $g(X^*)$
Kp1	(0, 1, 1, 1, 0, 0, 0, 1, 1, 1)	295	0
Kp2	(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1)	1024	−7
Kp3	(1, 1, 0, 1)	35	−2
Kp4	(0, 1, 0, 1)	23	0
Kp5	(0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1)	481.0694	−20.0392
Kp6	(0, 0, 1, 0, 1, 1, 1, 1, 0, 0)	50	0
Kp7	(1, 0, 0, 1, 0, 0, 0)	107	0
Kp8	(1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0)	9767	−232
Kp9	(1, 1, 1, 1, 0)	130	−20
Kp10	(1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1)	1025	−8
Kp11	(1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0)	3119	0
Kp12	(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1)	26559	−1

Table 3: The parameters of the high-dimensional 0-1 knapsack problems.

No.	Dim.	Capacity	maxFEs
Kp13	200	1500	15 000
Kp14	300	1700	20 000
Kp15	500	2000	20 000
Kp16	800	5000	30 000
Kp17	1000	10000	30 000
Kp18	1200	14000	40 000
Kp19	1500	16000	40 000
Kp20	2000	22000	50 000

Subsequently, DGHS is compared with GHS [35] and NGHS [24]. To make a fair comparison, the number of maximum improvisations, namely, the maximum number of function evaluations (maxFEs), is set to 1000 for all the approaches on the twelve benchmark instances listed in Table 1. For the other eight test instances with high dimensions, the settings of maxFEs are given in Table 3. In addition, the other specific parameters of algorithms are given as follows.

(i) For GHS, HMS is set to 5. The other control parameters are set as follows: HMCR = 0.9, $\text{PAR}_{\min} = 0.01$, and $\text{PAR}_{\max} = 0.99$. The settings are the same as those used in [35]. In order to handle the constraints, the penalty coefficient δ is set as the value used in [24], that is, 10^{20}.

(ii) For NGHS, HMS = 5, the mutation probability $p_m = 2/N$, and the penalty coefficient δ is set to 10^{20}, which are the same as those utilized in [24].

(iii) For DGHS, the harmony memory size HMS is also set to 5. And the other parameters are set as follows: $\text{HMCR}_{\max} = 0.95$, $\text{HMCR}_{\min} = 0.3$, and PAR = 0.75.

For all experiments, we use the aforementioned parameter settings unless a change is mentioned. Furthermore, in our experiments, each test problem is run over 50 independent times.

5.2. Comparison among GHS, NGHS, and DGHS. In this subsection, a variety of 0-1 knapsack problems with different scales are considered to investigate the performance of DGHS. And it is compared with GHS and DGHS.

First, experiments tested on twelve knapsack problems with small and median sizes are conducted. And the corresponding results are presented in Table 4 in terms of the best, worst, median, mean, standard deviation (Std.), and success rate (SR) of the solutions achieved in the 50 independent runs by each algorithm.

From Table 4, it can be seen that the DGHS can find global optimal values on the ten benchmark instances (Kp1–Kp10) with SR = 100%. NGHS can find global optima on the five test instances, that is, Kp1, Kp3, Kp4, Kp7, and Kp9. GHS can find global optima on the only three knapsack problems, that is, Kp3, Kp4, and Kp9. On the medium-scale knapsack problems Kp11 and Kp12, the success rate (SR) obtained by DGHS is 82% (41/50) for the test instance Kp11; SR obtained by DGHS is 32% (16/50) on the benchmark instance Kp12. Both GHS and NGHS fail to find global optima on the two instances. However, NGHS performs better than GHS on the two instances in terms of the best, worst, median, mean, and standard deviation of solutions. For the sake of convenience, later comparisons between NGHS and DGHS are conducted with larger values of maxFEs. On the knapsack problem Kp11, DGHS can find global optima with SR = 100% when maxFEs = 5*e*3. But NGHS still fails to find global optimal value. On the test problem Kp12, the success rate (SR) obtained by DGHS accordingly increases with the increase of maxFEs. Especially, DGHS can find the best known value 26559 with SR = 100% when the number of

Table 4: Comparison among three algorithms on solving 0-1 knapsack problems with small or median sizes.

No.	Dim.	maxFEs	Approach	Best	Worst	Median	Mean	Std.	SR
Kp1	10	1*e*3	GHS	295	293	295	294.9	0.3136	48/50
			NGHS	295	295	295	295	0	50/50
			DGHS	295	295	295	295	0	50/50
Kp2	20	1*e*3	GHS	1024	1018	1024	1023.7	1.1876	48/50
			NGHS	1024	995	1024	1018.6	8.0585	28/50
			DGHS	1024	1024	1024	1024	0	50/50
Kp3	4	1*e*3	GHS	35	35	35	35	0	50/50
			NGHS	35	35	35	35	0	50/50
			DGHS	35	35	35	35	0	50/50
Kp4	4	1*e*3	GHS	23	23	23	23	0	50/50
			NGHS	23	23	23	23	0	50/50
			DGHS	23	23	23	23	0	50/50
Kp5	15	1*e*3	GHS	481.0694	466.3380	481.0694	479.8091	3.3263	42/50
			NGHS	481.0694	437.9345	481.0694	479.1293	6.5594	41/50
			DGHS	481.0694	481.0694	481.0694	481.0694	0	50/50
Kp6	10	1*e*3	GHS	52	52	52	52	0	50/50
			NGHS	52	51	52	51.9400	0.2399	47/50
			DGHS	52	52	52	52	0	50/50
Kp7	7	1*e*3	GHS	107	105	107	106.8	0.6060	45/50
			NGHS	107	107	107	107	0	50/50
			DGHS	107	107	107	107	0	50/50
Kp8	23	1*e*3	GHS	9766	9744	9756.5	9756.2	5.3816	0/50
			NGHS	9767	9760	9765	9764.3	2.2950	15/50
			DGHS	9767	9767	9767	9767	0	50/50
Kp9	5	1*e*3	GHS	130	130	130	130	0	50/50
			NGHS	130	130	130	130	0	50/50
			DGHS	130	130	130	130	0	50/50
Kp10	20	1*e*3	GHS	1025	1019	1025	1024.1	2.1030	43/50
			NGHS	1025	979	1025	1020.8	8.2048	31/50
			DGHS	1025	1025	1025	1025	0	50/50
Kp11	50	1*e*3	GHS	2985	2837	2935	2932.5	35.9802	0/50
			NGHS	3087	2943	3002	3006.5	29.2779	0/50
			DGHS	3119	3114	3119	3118.7	0.9355	41/50
		5*e*3	GHS	3073	2955	3022	3018.9	28.1109	0/50
			NGHS	3104	3039	3077	3076.7	16.0329	0/50
			DGHS	3119	3119	3119	3119	0	50/50
Kp12	100	1*e*3	GHS	24551	22867	23630.5	23648.6	382.0742	0/50
			NGHS	25817	24209	24974	25028	361.3838	0/50
			DGHS	26559	26534	26541	26544	10.6246	16/50
		5*e*3	GHS	25448	24343	24891	24881	262.5590	0/50
			NGHS	26418	25686	26102	26102	165.5934	0/50
			DGHS	26559	26536	26559	26553	8.6175	33/50
		1*e*4	GHS	26079	24806	25296	25304	219.1038	0/50
			NGHS	26428	25896	26282	26264	105.0596	0/50
			DGHS	26559	26547	26559	26556	5.0214	39/50
		2*e*4	GHS	26091	25151	25685	25683	223.8487	0/50
			NGHS	26489	26242	26366	26367	59.8389	0/50
			DGHS	26559	26547	26559	26558	3.2886	46/50
		3*e*4	GHS	26172	25379	25816	25833	166.5229	0/50
			NGHS	**26559**	26265	26424	26420	63.9882	**1/50**
			DGHS	26559	26547	26559	26558.5	2.3754	48/50
		4*e*4	GHS	26326	25643	25947	25954	142.3878	0/50
			NGHS	**26547**	26354	26441	26445	46.2120	**0/50**
			DGHS	26559	26559	26559	26559	0	50/50

Table 5: Comparison among GHS, NGHS, and DGHS on the high-dimensional 0-1 knapsack problems.

No.	Dim.	maxFEs	Index	GHS	NGHS	DGHS
Kp13	200	1.5*e*4	Best	10306	10887	11025
			Worst	9950	10756	11019
			Median	10148.5	10840	11021.5
			Mean	10139.2	10834.3	11021.6
			Std.	91.9592	34.6055	2.1055
			t-test	1	1	/
Kp14	300	2*e*4	Best	12608	13878	14086
			Worst	12126	13614	14080
			Median	12396	13746.5	14085
			Mean	12396.4	13752.9	14084.8
			Std.	112.1964	63.0862	1.3756
			t-test	1	1	/
Kp15	500	2*e*4	Best	15270	18271	18925
			Worst	14558	17735	18916
			Median	14886.5	18007	18924
			Mean	14904.7	18014.4	18923.3
			Std.	157.0984	132.3975	2.1801
			t-test	1	1	/
Kp16	800	3*e*4	Best	34245	38703	39691
			Worst	33351	38284	39691
			Median	33871.5	38489.5	39691
			Mean	33835.1	38487.5	39691
			Std.	191.8773	97.0605	0
			t-test	1	1	/
Kp17	1000	3*e*4	Best	60906	65078	66109
			Worst	60349	64332	66106
			Median	60661.5	64723.5	66109
			Mean	60668.2	64715.4	66108.6
			Std.	129.2363	150.1521	0.6966
			t-test	1	1	/
Kp18	1200	4*e*4	Best	78067	86099	86771
			Worst	76093	85657	86771
			Median	77241.5	85899	86771
			Mean	77185.3	85901.1	86771
			Std.	491.5674	88.4639	0
			t-test	1	1	/
Kp19	1500	4*e*4	Best	98072	104199	105797
			Worst	94516	103428	105794
			Median	95967	103802.5	105797
			Mean	95974.3	103798.9	105796.6
			Std.	807.3958	172.6269	0.7530
			t-test	1	1	/
Kp20	2000	5*e*4	Best	124049	138509	140710
			Worst	118599	137754	140704
			Median	121535.5	138170	140709
			Mean	121417.8	138165.2	140709
			Std.	102.6351	170.3528	1.3559
			t-test	1	1	/

Table 6: Comparison among DGHS and IGHS together with IGHS-II on ten knapsack benchmarks.

No.	maxFEs	Index	IGHS			IGHS-II			DGHS		
			Mean	Std.	SR	Mean	Std.	SR	Mean	Std.	SR
Kp1	1000	$D = 10$	238.18	38.5289	2/50	294.16	0.3703	8/50	295	0	50/50
		Rank	3			2			1		
Kp2	1000	$D = 20$	936.36	52.4446	1/50	1018.48	1.6442	4/50	1024	0	50/50
		Rank	3			2			1		
Kp3	1000	$D = 4$	32.80	2.8500	25/50	35	0	50/50	35	0	50/50
		Rank	3			1			1		
Kp4	1000	$D = 4$	21.08	2.1931	20/50	21.02	2.1043	18/50	23	0	50/50
		Rank	2			3			1		
Kp5	1000	$D = 15$	386.88	63.4233	8/50	481.0694	0	50/50	481.0694	0	50/50
		Rank	3			1			1		
Kp6	1000	$D = 10$	48.96	2.9413	16/50	52	0	50/50	52	0	50/50
		Rank	3			1			1		
Kp7	1000	$D = 7$	95.10	9.1032	2/50	104.54	2.1591	17/50	107	0	50/50
		Rank	3			2			1		
Kp8	1000	$D = 23$	9695.42	55.4923	1/50	9760.72	6.9400	26/50	9767	0	50/50
		Rank	3			2			1		
Kp9	1000	$D = 5$	120.46	9.0649	21/50	130	0	50/50	130	0	50/50
		Rank	3			1			1		
Kp10	1000	$D = 20$	955.92	49.5991	1/50	1019.36	1.4393	3/50	1025	0	50/50
		Rank	3			2			1		
Average rank			2.9			1.7			1		
Overall rank			3			2			1		

maximum improvisations maxFEs is set to 4e4. Yet NGHS cannot be able to find the best known solution. Although NGHS find the best known value 26559 one time out of fifty times when maxFEs = 3e4, the success rate (SR) obtained by NGHS is still zero when maxFEs = 4e4. This indicates that the performance of NGHS is unstable. In a word, DGHS is the best among the three algorithms according to the robustness and convergence performance of algorithms.

Second, eight high-dimensional knapsack problems are generated randomly to further testify the comprehensive performance of DGHS. The statistical results obtained in 50 independent runs by three algorithms are given in Table 5. Moreover, the t-test results of all the test instances with high dimensions are also given in Table 5, in which "1" indicates that the proposed DGHS is significantly better than its competitor (GHS or NGHS) at the level of significance $\alpha = 0.05$.

As can be seen from Table 5, GHS is obviously inferior to NGHS on all the test instances with high dimensions. And from the t-test results in Table 5, it is observed that DGHS outperforms significantly both GHS and NGHS on all the test problems. It is worth mentioning that the worst value of profit sum obtained by DGHS is even better than the best value of profit sum found by GHS and NGHS on all the test instances. In addition, the standard deviation of profit sum obtained by DGHS is also very small on each test problem, which indicates that the DGHS is robust. All these indicate that DGHS has an overwhelming advantage against the other two algorithms on solving the knapsack problems with large scales.

5.3. Further Comparison. Recently, El-Abd had proposed an improved global-best harmony search algorithm, named IGHS, which achieves a better performance for solving real parameter optimization problems [30]. In order to further testify the performance of DGHS, DGHS is further compared with two variants of IGHS for solving 0-1 knapsack problems. For convenience, the first version of IGHS with penalty function is called IGHS. Another version of IGHS with a greedy initialization together with the two-phase repair operator is called IGHS-II. That is, IGHS can be used to handle the knapsack problems by using a penalty function scheme as recommended in [24]. IGHS-II can also be used to solve the knapsack problems through integrating two-phase repair operator and a greedy initialization scheme, which are the same as those employed in DGHS.

For a fair comparison, the maximum number of function evaluations (maxFEs) is set to 1000 for all experiments. In addition, the other parameters of the above two variants (IGHS and IGHS-II) are the same as those of the original IGHS except maxFEs. For IGHS and IGHS-II, HMS = 5, HMCR = 0.99, $\mathrm{PAR}_{\min} = 0.01$, $\mathrm{PAR}_{\max} = 0.99$, $\mathrm{bw}_{\min} = 0.0001$, and $\mathrm{bw}_{\max} = (U_j - L_j)/20$. Meanwhile, the other specific parameters of DGHS are the same as the aforementioned. And ten knapsack problems, that is,

Kp1–Kp10, are employed here to perform the experiment. Subsequently, each case of all compared algorithms is run 50 times independently and all experimental results are listed in Table 6.

From Table 6, it can be seen that IGHS is capable of finding the global optimum for each knapsack problem (Kp1–Kp10) in terms of SR (success rate). Especially, the SR obtained by IGHS on Kp3, Kp4, and Kp9 is over 40%, respectively. Yet IGHS-II outperforms IGHS on nine benchmarks, that is, Kp1, Kp2, Kp3, Kp5, Kp6, Kp7, Kp8, Kp9, and Kp10. What is more, the SR obtained by IGHS-II on Kp4 is also very close to that found by IGHS. All these indicate that IGHS works well on knapsack problems and the two-phase repair operator is active. It is worth noting that DGHS achieves the corresponding global optimum for each of ten knapsack problems with 100% success rate, which shows that DGHS is obviously superior to IGHS and DGHS is better than or at least similar to IGHS-II on all benchmarks. And the two-phase repair operator and greedy initialization scheme are all used in IGHS-II and DGHS, respectively. In view of this, DGHS itself is also active. That is, DGHS can work better than IGHS for solving 0-1 knapsack problems. According to the overall rank in Table 6, it is observed that DGHS takes the first place when compared against IGHS and IGHS-II.

6. Conclusion

In this work, a novel discrete global-best harmony search algorithm, called DGHS, is proposed through introducing some modifications, such as an initialization based on a greedy scheme used to improve the solution quality of initial harmony memory, a novel binary coded global harmony search algorithm based on intuitive cognition of improvisation process for easily performing discrete operation and effectively taking advantage of the guiding information of the best harmony, and two-phase repair operator used to repair an infeasible harmony vector and to further improve a feasible solution. The experiments tested on twenty knapsack problems are conducted. The experimental results reveal that DGHS outperforms both GHS and NGHS in most cases. Thus, the proposed DGHS can be considered as an elitist alternative for solving the 0-1 knapsack problems with different scales.

In the future, it is desired to be further applied to solve some real life problems, like warehouse location, production planning, portfolio optimization, and so on.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (Grants nos. 61064012, 61164003, and 61364026), the Youth Science Foundation of Lanzhou Jiaotong University (Grant no. 2012029), and the Science and Technology Foundation of Lanzhou Jiaotong University (Grant no. ZC2012005). The authors also wish to thank the referees for their constructive comments and suggestions.

References

[1] V. Chvata, "Hard knapsack problems," *Operations Research*, vol. 28, no. 6, pp. 1402–1411, 1980.

[2] C. S. Chung, M. S. Hung, and W. Rom, "A hard knapsack problem," *Naval Research Logistics*, vol. 35, pp. 85–98, 1988.

[3] G. È. Cornuèjols and M. Dawande, "A class of hard small 0-1 programs," *INFORMS Journal on Computing*, vol. 11, no. 2, pp. 205–210, 1999.

[4] A. V. Carbot, "An enumeration algorithm for knapsack problems," *Operations Research*, vol. 18, no. 2, pp. 306–311, 1970.

[5] P. C. Gilmore and R. E. Gomory, "The theory and computation of knapsack functions," *Operations Research*, vol. 14, no. 6, pp. 1045–1074, 1966.

[6] S. Martello, D. Pisinger, and P. Toth, "Dynamic programming and strong bounds for the 0-1 knapsack problem," *Management Science*, vol. 45, no. 3, pp. 414–424, 1999.

[7] J. K. Peter, "A branch and bound algorithm for the knapsack problem," *Management Science*, vol. 13, no. 6, pp. 723–735, 1967.

[8] D. Pisinger, "Where are the hard knapsack problems?" *Computers and Operations Research*, vol. 32, no. 9, pp. 2271–2284, 2005.

[9] S. Dash, "Exponential lower bounds on the lengths of some classes of branch-and-cut proofs," *Mathematics of Operations Research*, vol. 30, no. 3, pp. 678–700, 2005.

[10] G. J. Koehler, "Minimal equivalent binary knapsack inequalities," *INFORMS Journal on Computing*, vol. 23, no. 3, pp. 425–429, 2010.

[11] Y. C. He, K. Q. Liu, C. J. Zhang, and W. Zhang, "Greedy genetic algorithm for solving kanpsack problems and its applications," *Computer Engineering and Design*, vol. 28, no. 11, pp. 2655–2658, 2007.

[12] J. C. Bansal and K. Deep, "A modified binary particle swarm optimization for knapsack problems," *Applied Mathematics and Computation*, vol. 218, no. 22, pp. 11042–11061, 2012.

[13] T. K. Truong, K. Li, and Y. Xu, "Chemical reaction optimization with greedy strategy for the 0-1 knapsack problem," *Applied Soft Computing*, vol. 13, no. 4, pp. 1774–1780, 2013.

[14] A. Layeb, "A hybrid quantum inspired harmony search algorithm for 0-1 optimization problems," *Journal of Computational and Applied Mathematics*, vol. 253, pp. 14–25, 2013.

[15] L. Wang, R. Yang, Y. Xu et al., "An improved adaptive binary harmony search algorithm," *Information Sciences*, vol. 232, pp. 58–87, 2013.

[16] J. Roland, J. R. Figueira, and Y. D. Smet, "The inverse {0, 1}-knapsack problem: theory, algorithms and computational experiments," *Discrete Optimization*, vol. 10, no. 2, pp. 181–192, 2013.

[17] A. Fréville, "The multidimensional 0-1 knapsack problem: an overview," *European Journal of Operational Research*, vol. 155, no. 1, pp. 1–21, 2004.

[18] A. Billionnet and É. Soutif, "An exact method based on lagrangian decomposition for the 0-1 quadratic knapsack problem," *European Journal of Operational Research*, vol. 157, no. 3, pp. 565–575, 2004.

[19] A. Billionnet and É. Soutif, "Using a mixed integer programming tool for solving the 0-1 quadratic knapsack problem,"

INFORMS Journal on Computing, vol. 16, no. 2, pp. 188–197, 2004.

[20] L. Jourdan, M. Basseur, and E. G. Talbi, "Hybridizing exact methods and metaheuristics: a taxonomy," *European Journal of Operational Research*, vol. 199, no. 3, pp. 620–629, 2009.

[21] V. Boyer, M. Elkihel, and D. El Baz, "Heuristics for the 0-1 multidimensional knapsack problem," *European Journal of Operational Research*, vol. 199, no. 3, pp. 658–664, 2009.

[22] S. Siva Sathya and S. Kuppuswami, "Gene silencing—a genetic operator for constrained optimization," *Applied Soft Computing Journal*, vol. 11, no. 8, pp. 5801–5808, 2011.

[23] M. Vasquez and Y. Vimont, "Improved results on the 0-1 multidimensional knapsack problem," *European Journal of Operational Research*, vol. 165, no. 1, pp. 70–81, 2005.

[24] D. Zou, L. Gao, S. Li, and J. Wu, "Solving 0-1 knapsack problem by a novel global harmony search algorithm," *Applied Soft Computing Journal*, vol. 11, no. 2, pp. 1556–1564, 2011.

[25] V. C. B. Camargo, L. Mattiolli, and F. M. B. Toledo, "A knapsack problem as a tool to solve the production planning problem in small foundries," *Computers and Operations Research*, vol. 39, no. 1, pp. 86–92, 2012.

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

[27] S. Das, A. Mukhopadhyay, A. Roy, A. Abraham, and B. K. Panigrahi, "Exploratory power of the harmony search algorithm: analysis and improvements for global numerical optimization," *IEEE Transactions on Systems, Man, and Cybernetics B: Cybernetics*, vol. 41, no. 1, pp. 89–106, 2011.

[28] P. Yadav, R. Kumar, S. K. Panda, and C. S. Chang, "An intelligent tuned harmony search algorithm for optimisation," *Information Sciences*, vol. 196, pp. 47–72, 2012.

[29] B. Wu, C. Qian, W. Ni et al., "Hybrid harmony search and artificial bee colony algorithm for global optimization problems," *Computers & Mathematics with Applications*, vol. 64, no. 8, pp. 2621–2634, 2012.

[30] M. El-Abd, "An improved global-best harmony search algorithm," *Applied Mathematics and Computation*, vol. 222, pp. 94–106, 2013.

[31] D. Zou, L. Gao, S. Li, J. Wu, and X. Wang, "A novel global harmony search algorithm for task assignment problem," *Journal of Systems and Software*, vol. 83, no. 10, pp. 1678–1688, 2010.

[32] D. Zou, L. Gao, J. Wu, and S. Li, "Novel global harmony search algorithm for unconstrained problems," *Neurocomputing*, vol. 73, no. 16-18, pp. 3308–3318, 2010.

[33] D. Zou, L. Gao, J. Wu, S. Li, and Y. Li, "A novel global harmony search algorithm for reliability problems," *Computers and Industrial Engineering*, vol. 58, no. 2, pp. 307–316, 2010.

[34] L. Wang, Q. K. Pan, and M. F. Tasgetiren, "A hybrid harmony search algorithm for the blocking permutation flow shop scheduling problem," *Computers and Industrial Engineering*, vol. 61, no. 1, pp. 76–83, 2011.

[35] M. G. H. Omran and M. Mahdavi, "Global-best harmony search," *Applied Mathematics and Computation*, vol. 198, no. 2, pp. 643–656, 2008.

[36] M. Mahdavi, M. Fesanghary, and E. Damangir, "An improved harmonysearch algorithm for solving optimization problems," *Applied Mathematics and Computation*, vol. 188, no. 2, pp. 1567–1579, 2007.

[37] B. Alatas, "Chaotic harmony search algorithms," *Applied Mathematics and Computation*, vol. 216, no. 9, pp. 2687–2699, 2010.