

Article

A Heuristics-Guided Simplified Discrete Harmony Search Algorithm for Solving 0-1 Knapsack Problem

Fuyuan Zheng ^{1,2} , Kanglong Cheng ^{1,2}, Kai Yang ^{1,2}, Ning Li ^{1,2}, Yu Lin ^{1,2,*}  and Yiwen Zhong ^{1,2} 

¹ College of Computer and Information Sciences, Fujian Agriculture and Forestry University, Fuzhou 350002, China; 12311093003@fafu.edu.cn (F.Z.); 12411093017@fafu.edu.cn (K.C.); 52311049032@fafu.edu.cn (K.Y.); 52411049028@fafu.edu.cn (N.L.); yiwzhong@fafu.edu.cn (Y.Z.)

² Key Laboratory of Smart Agriculture and Forestry, Fujian Province University, Fuzhou 350002, China

* Correspondence: linyu@fafu.edu.cn

Abstract: The harmony search (HS) algorithm is a novel metaheuristic which has been widely used to solve both continuous and discrete optimization problems. In order to improve the performance and simplify the implementation of the HS algorithm for solving the 0-1 knapsack problem (0-1KP), this paper proposes a heuristics-guided simplified discrete harmony search (SDHS) algorithm which does not use random search operator and has only one intrinsic parameter, harmony memory size. The SDHS algorithm uses a memory consideration operator to construct a feasible solution, and then the constructed solution is further enhanced by a solution-level pitch adjustment operator. Two heuristics, the profit-weight ratio of an item and the profit of an item, are used to greedily guide the memory consideration operator and the solution-level pitch adjustment operator, respectively. In the memory consideration operator, items are considered in non-ascending order of profit-weight ratio assigned from the harmony memory. In the solution-level pitch adjustment operator, items not in the knapsack are attempted to be selected in non-ascending order of profit. The SDHS algorithm outperforms several state-of-the-art algorithms, with an average improvement of 0.55% in the quality of solutions on large problem instances.

Keywords: harmony search; 0-1 knapsack problem; memory consideration; solution-level pitch adjustment; profit-weight ratio; profit



Academic Editor: Roberto Montemanni

Received: 4 March 2025

Revised: 28 April 2025

Accepted: 9 May 2025

Published: 19 May 2025

Citation: Zheng, F.; Cheng, K.; Yang, K.; Li, N.; Lin, Y.; Zhong, Y. A Heuristics-Guided Simplified Discrete Harmony Search Algorithm for Solving 0-1 Knapsack Problem. *Algorithms* **2025**, *18*, 295. <https://doi.org/10.3390/a18050295>

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The knapsack problem (KP) is a classical combinatorial optimization problem with diverse applications across numerous fields. Over time, KP has evolved into a family of variants, including the 0-1 Knapsack Problem (0-1KP), Multidimensional Knapsack Problem (MKP), Quadratic Knapsack Problem, and the Generalized Knapsack Sharing Problem. Among these, the 0-1KP, as the foundational formulation of KP, plays a critical role in both theoretical and applied contexts. For instance, it has been utilized in real estate maintenance optimization [1], energy consumption minimization [2], cargo loading optimization [3], cryptographic systems [4], network security interdiction [5], and portfolio selection strategies [6]. These applications underscore the versatility of the 0-1KP in addressing complex resource allocation challenges under constraints.

The 0-1KP is NP-hard, necessitating the development of advanced metaheuristics to approximate optimal solutions [7]. Prominent examples of these metaheuristics include the cuckoo search algorithm [8–10], firefly algorithm [10], social spider algorithm [11], cohort intelligence algorithm [12], monkey algorithm [13], wolf-inspired algorithms [14–18], bat algorithm [19], wisdom of artificial crowds algorithm [20], symbiotic

organisms search algorithm [21], elite opposition-flower pollination algorithm [22], chemical reaction optimization algorithm [23,24], genetic algorithm [25,26], monarch butterfly optimization algorithm [27–29], particle swarm optimization algorithm [30], whale optimization algorithm [31], shuffled frog leaping algorithm [32,33], slime mold algorithm [34], rice optimization algorithm [35], Differential Evolution algorithm [36], Quadratic Interpolation Optimization algorithm [36], Manta Ray Foraging Optimization algorithm [36], Artificial Jellyfish Search algorithm [37], Kepler Optimization Algorithm [38], harmony search algorithm [39–46], local search algorithm [47–49], etc. Most of the aforementioned algorithms require additional transfer functions to obtain binary solutions from continuous search space, use a hybrid of multiple metaheuristics, use advanced computation technologies, use enhanced operators, and/or use extra operators. Take the five cuckoo search algorithms [8–10,50,51] as an example: (1) all those five algorithms use the Sigmoid function to transfer a real number into binary; (2) ref. [50] uses the genetic mutation operation to prevent it from being trapped in the local optimum; (3) ref. [51] uses a hybrid of the cuckoo search algorithm and shuffled frog-leaping algorithm; (4) ref. [9] uses a hybrid of the cuckoo search algorithm and harmony search; and (5) ref. [10] uses the stochastic hill-climbing technique to improve its intensification ability. Although those advanced strategies are helpful to obtaining good performance, they give rise to extra complexity also.

The harmony search (HS) algorithm [52] is a kind of construction-based metaheuristic which was proposed to solve both continuous and combinatorial optimization problems. Inspired by the music improvisation process, the HS algorithm uses three operators, memory consideration operator, random search operator, and pitch adjustment operator, to construct a solution [53]. Since its publication, many improved HS algorithms have been proposed and successfully applied to many real-world optimization problems [54–59]. Several HS algorithms [39–46] have been successfully proposed to solve the 0-1KP. As stated in [60] and [61], the performance of the HS algorithm is remarkably affected by its parameters, and it is annoying to fine-tune those parameters. In order to further improve the performance and simplify the implementation of the HS algorithm for solving the 0-1KP, this paper proposes a heuristics-guided simplified discrete harmony search (SDHS) algorithm which does not use random search operator and has only one intrinsic parameter, harmony memory size (hms). The proposed SDHS algorithm uses the memory consideration operator to construct a feasible solution, and then the constructed solution is enhanced by a novel solution-level pitch adjustment operator. Two heuristics, the profit–weight ratio (p_i/w_i) of an item and the profit (p_i) of an item, are used to greedily guide the memory consideration operator and the solution-level pitch adjustment operator, respectively. In the memory consideration operator, items are considered in non-ascending order of p_i/w_i to be assigned values from the harmony memory (HM). In the solution-level pitch adjustment operator, items not in the knapsack are tried to be selected in non-ascending order of p_i . Both the memory consideration operator and the solution-level pitch adjustment operator are carried out under the restriction of capacity constraint (2). Three experiments are performed to analyze the effect of the parameter hms on the performance, the convergence, and the diversity of the SDHS algorithm. Comparative experiments, conducted on three sets of large-scale 0-1KP instances, have shown that the SDHS algorithm performs better than other state-of-the-art metaheuristics.

The rest of this paper is organized as follows. Section 2 provides a short description of the basic HS algorithm and the existing HS algorithms for the 0-1KP. Section 3 presents the proposed SDHS algorithm for the 0-1KP. Section 4 analyzes the effect of the strategies and the parameter hms on the performance of the SDHS algorithm. Section 5 compares the performance of the SDHS algorithm with other state-of-the-art metaheuristics on three sets of large-scale 0-1KP instances. Finally, in Section 6, we summarize our study.

2. Preliminaries

2.1. Problem Formulation

Given a knapsack with maximum capacity W and a set of n items where each item i has a profit p_i and a weight w_i , the objective of the 0-1KP is to pack the knapsack with items such that a maximum profit is obtained under the constraint that the total weight of the items in the knapsack does not exceed the capacity of the knapsack. Mathematically, the 0-1KP can be modeled as the following 0-1 integer programming problem:

$$\max f(\mathbf{x}) = \sum_{i=1}^n p_i x_i \quad (1)$$

$$s.t. \quad \sum_{i=1}^n w_i x_i \leq W \quad (2)$$

$$x_i \in \{0, 1\} \quad (3)$$

where x_i indicates whether item i is packed in the knapsack. If x_i is equal to 1, then item i is packed in the knapsack; otherwise, it is discarded. Constraint (2) is the capacity constraint which guarantees that the total weight of the items in the knapsack is not greater than the knapsack's capacity, and constraint (3) is the whole constraint which guarantees that no item is broken.

2.2. The Harmony Search Algorithm

The HS algorithm is a metaheuristic that mimics the improvisation process of music players to obtain better harmony. The basic HS algorithm uses three operators, i.e., the memory consideration operator, the pitch adjustment operator, and the random search operator, to cooperatively construct a new solution.

Given a solution represented as a vector x with n elements, Algorithm 1 describes the steps of the basic HS algorithm, in which lines 4 to 13 are used to construct a new solution x^{new} until the stop condition is met (i.e., the maximum number of iterations or no improvement for certain iterations). Line 6, line 8, and line 11 are the memory consideration operator, the pitch adjustment operator, and the random search operator, respectively. In line 14, the newly produced solution x^{new} is used to update the harmony memory if necessary.

Algorithm 1 The basic HS algorithm.

```

1: Initialize the parameters of the HS algorithm
2: Initialize the harmonies (solutions) in harmony memory
3: while Stop condition is not met do
4:   for each dimension  $j$  from 1 to  $n$  do
5:     if  $\text{rand}() < \text{hmcr}$  then
6:       // Memory Consideration Operator
7:        $x_j^{new} = x_j^i$ , where  $i$  is randomly selected from 1 to  $\text{hms}$ 
8:       if  $\text{rand}() < \text{par}$  then
9:         // Pitch Adjustment Operator
10:        Adjust  $x_j^{new}$ 
11:       end if
12:     else
13:       // Random Search Operator
14:       Randomly create a value for  $x_j^{new}$ 
15:     end if
16:   end for
17:   Use  $x^{new}$  to update harmony memory
18: end while

```

The HS algorithm stores several historical solutions in the harmony memory (HM), which can be represented by a matrix as Equation (4):

$$HM = \begin{pmatrix} x_1^1 & x_2^1 & \dots & x_{n-1}^1 & x_n^1 \\ x_1^2 & x_2^2 & \dots & x_{n-1}^2 & x_n^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_1^{hms-1} & x_2^{hms-1} & \dots & x_{n-1}^{hms-1} & x_n^{hms-1} \\ x_1^{hms} & x_2^{hms} & \dots & x_{n-1}^{hms} & x_n^{hms} \end{pmatrix} \quad (4)$$

where hms is a parameter which represents the size of the harmony memory, and n is the number of decision variables of the problem at hand. Except for the hms , the basic HS algorithm has two additional intrinsic parameters, harmony memory consideration rate ($hmcr$) and pitch adjustment rate (par), which control the probability of selecting a solution component from the harmony memory and the probability of enhancing the selected component, respectively.

The basic HS algorithm is a general framework which may be implemented as continuous HS (CHS) algorithm or discrete HS (DHS) algorithm. For the CHS algorithm, another parameter bw , which represents an arbitrary distance bandwidth, is introduced to implement the pitch adjustment operator as the following:

$$x_j^{new} = x_j^{new} \pm rand() \times bw \quad (5)$$

In order to implement random search operator for the CHS algorithm, the value of x_j^{new} can be randomly created as the following:

$$x_j^{new} = x_{j,min} + rand() \times (x_{j,max} - x_{j,min}) \quad (6)$$

where $x_{j,min}$ and $x_{j,max}$ represent the minimum value and the maximum value of the j th dimension, respectively. For the implementation of the DHS algorithm, the random search operator can simply select a new value from all possible values for x_j^{new} , and the pitch adjustment operator may be abolished or redesigned according to the characteristics of the optimization problem at hand.

Both the CHS algorithm and the DHS algorithm can be used to solve the combinatorial optimization problem. In the field of the 0-1KP, the novel global harmony search (NGHS) algorithm [39] and the harmony search algorithm based on teaching–learning strategies (HSTL) [43] belong to the CHS algorithm. In order to use the CHS algorithm for the 0-1KP, a mapping strategy must be specified to map a real number into 0 or 1. In the NGHS algorithm, because the position updating equations are defined for real numbers in the range of 0 to 1, every variable is replaced by the nearest integer to obtain 0 or 1. Same strategy is used by the HSTL algorithm to obtain binary solution from continuous variables. The adaptive binary harmony search (ABHS) algorithm [41], the binary modified harmony search (BMHS) algorithm [42], the discrete global best harmony search (DGHS) algorithm [44], and the simplified binary harmony search (SBHS) algorithm [45] belong to the DHS algorithm. We introduce these DHS algorithms in detail in the next subsection.

2.3. Discrete HS Algorithms for the 0-1KP

In the ABHS algorithm [41], two strategies, the bit selection strategy and the individual selection strategy, are designed to implement the memory consideration operator. The bit

selection strategy implements the classical memory consideration operator, and the individual selection strategy implements a new memory consideration operator as the following:

$$x_j^{new} = \begin{cases} x_j^p, & \text{if } rand() < hmcr \\ round(rand()), & \text{otherwise} \end{cases} \quad (7)$$

where x^p is a random harmony which is fixed for all items of x^{new} , and the *round* function rounds a number to the nearest integer. The random search operator of the ABHS algorithm randomly selects a new value (0 or 1) for x_j^{new} . The pitch adjustment operator of the ABHS algorithm is designed to learn from the global best harmony with probability *par*.

In the BMHS algorithm [42], the pitch adjustment operator is designed to learn from the global best harmony or to produce a new value randomly as the following:

$$x_j^{new} = \begin{cases} x_j^{best}, & \text{if } rand() < 0.5 \\ round(rand()), & \text{otherwise} \end{cases} \quad (8)$$

where x^{best} is the best harmony in the harmony memory. The random search operator of the BMHS algorithm randomly selects a new value (0 or 1) for x_j^{new} .

In order to enhance the local search ability of HS algorithm for the 0-1KP, the DGHS algorithm [44] heavily uses the best harmony to guide the improvisation of new solution. In the DGHS algorithm, a new solution is improvised as the following:

$$x_j^{new} = \begin{cases} x_j^{best}, & \text{if } rand() < hmcr \\ x_j^i, & \text{otherwise} \end{cases} \quad (9)$$

where x^{best} is the best harmony in the harmony memory, and x^i is randomly selected from the harmony memory. Furthermore, the x_j^{new} , which is not selected from x_j^{best} , is flipped with probability *par*.

In order to appropriately balance the intensification and diversification of the HS algorithm, the SBHS algorithm [45] proposes an ingenious improvisation scheme as the following:

$$x_j^{new} = x_j^{r1} + (-1)^{x_j^{r1}} \times (x_j^{best} \oplus x_j^{r2}), j \in \{1, 2, \dots, n\} \quad (10)$$

where $r1$ and $r2$ are two distinct random integers between 1 and *hms* for each dimension j . Equation (10) combines the memory consideration and the pitch adjustment to improvise each bit of a new harmony, so the SBHS algorithm does not need parameter *par*. On the right-hand side of Equation (10), the first term represents the memory consideration operator, and the second term represents the pitch adjustment operator. The exclusive OR of x_j^{best} and x_j^{r2} determines the value of x_j^{new} . If $x_j^{best} \oplus x_j^{r2}$ is equal to 0, then x_j^{r1} is assigned to x_j^{new} directly; otherwise, $1 - x_j^{r1}$ is assigned to x_j^{new} . The SBHS algorithm has only two parameters, i.e., *hms* and *hmcr*, which greatly simplifies the fine-tuning of parameters.

3. Simplified Discrete Harmony Search Algorithm for the 0-1KP

The HS algorithm is a simple, effective, and flexible metaheuristic for both continuous and discrete optimization problems. Among the three components of the HS algorithm, the memory consideration operator, which learns from historical experience stored in the harmony memory, is the core component. The other two components, the pitch adjustment operator and the random search operator, are designed to enhance its intensification ability and enlarge its diversification, respectively.

The cooperation of these two components can obtain good balance between exploration and exploitation for the HS algorithm. Although the basic HS algorithm is both

simple and easy to implement, to design an HS algorithm with excellent performance for a specific problem is far from a simple task.

Based on the characteristics of the problem at hand, we must analyze the three components of the basic HS algorithm carefully to answer the following two questions: (1) whether a component is needed for the problem at hand, and (2) how to implement this component if it is needed. The 0-1KP has the following two characteristics. One is that each variable has only two possible values, i.e., 0 or 1. The other is that an item with a high profit–weight ratio or profit should be picked with high priority. Based on these two key characteristics of the 0-1 knapsack problem (0-1KP), the necessity and implementation details of the memory consideration operator, pitch adjustment operator, and random search operator are explored in the following subsections.

3.1. Discussions of the HS Algorithm

The main purpose of the random search operator is to maintain enough diversity for the HS algorithm to have an optimization ability even in the late stage of the search. To obtain this objective, the CHS algorithm uses Equation (6) to randomly create a new value for x_j^{new} . The 0-1KP is a discrete optimization problem where the value of x_j^{new} can only be 0 or 1. Consequently, $x_{j,min}$ and $x_{j,max}$ are 0 and 1, respectively. A possible implementation of the random search operator is to randomly select a value from $\{0,1\}$ for the x_j^{new} with probability $1-hmcr$, so both 0 and 1 can have a chance to be the value of x_j^{new} . As a result, Equation (6) can be rewritten as represented as Equation (11):

$$x_j^{new} = x_{j,min} + rand() \times (x_{j,max} - x_{j,min}) = 0 + rand() * (1 - 0) \in \{0,1\} \quad (11)$$

In the case of the 0-1KP, the HM maintains a repository of historical solutions. Let $\mathbf{HM}(j)$ denote the historical solutions corresponding to the j -th dimension. When the harmony memory contains a sufficient number of solutions, new values x_j^{new} can be directly retrieved from $\mathbf{HM}(j)$ as shown in Equation (12). This approach achieves comparable outcomes to the random search operator, thereby rendering the parameter $hmcr$ obsolete. Consequently, the random search operator is eliminated, and each x_j^{new} is consistently selected from the harmony memory:

$$x_j^{new} \leftarrow \mathbf{HM}(j) = \begin{pmatrix} x_j^1 \\ x_j^2 \\ \vdots \\ x_j^{hms-1} \\ x_j^{hms} \end{pmatrix} \quad (12)$$

The pitch adjustment operator of the standard HS aims to improve its local search ability. The classical pitch adjustment operator is implemented at the dimension level, and the binary solution space such as 0-1KP, bw for the pitch adjustment operator is 1, and thus x_j^{new} can be adjusted according to Equation (13):

$$x_j^{new} = (x_j^{new} \pm rand() \times bw) \in \{0,1\} \quad (13)$$

As noted by Wang et al. in [41], the pitch adjustment operator of the HS algorithm is degenerated in the binary space, necessitating the development of a new approach to enhance its intensification ability. In solving the 0-1KP, most metaheuristics employ a greedy optimization operator to refine solutions. Unlike the classical pitch adjustment operator, which operates at the dimension level, the greedy optimization functions at the solution level. Specifically, for a n -dimensional solution $x = (x_1, x_2, \dots, x_n)$, the new

solution x^{new} is generated by applying a greedy optimization operator $Opt(\cdot)$ as defined in Equation (14):

$$x^{new} = Opt(x) \quad (14)$$

Given that the greedy optimization operator substantially strengthens the intensification capability, the classical pitch adjustment operator is no longer required for the 0-1 knapsack problem (0-1KP) and is therefore omitted.

In the 0-1KP, the capacity constraint of the knapsack introduces an indirect competitive relationship among items. For instance, incorporating an item α with profit p_α and weight w_α may hinder the inclusion of another item β (with profit p_β and weight w_β) due to the limited capacity. To address this, we evaluate items based on their profit-to-weight ratio (pw) or individual profit, where pw is defined as Equation (15):

$$pw_j = \frac{p_j}{w_j} \quad (15)$$

Given that the primary objective of the 0-1KP is to maximize profit, the memory consideration operator processes items in non-ascending order of their profit-to-weight ratio or individual profit.

3.2. Description of the SDHS Algorithm

Accordingly, based on the aforementioned analysis, the random search operator is omitted from our SDHS algorithm. Consequently, the memory consideration operator prioritizes items based on their profit-to-weight ratio pw , while the pitch adjustment operator is executed at the solution level. In the subsequent subsections, we detail the memory consideration operator, the solution-level pitch adjustment mechanism, and the comprehensive framework of the SDHS algorithm. And the pseudo code of the proposed SDHS algorithm for 0-1KP is provided in Algorithm 2. Figure 1 shows the flowchart of the SDHS algorithm.

Algorithm 2 Pseudo code of the SDHS algorithm for the 0-1KP.

Require: m, hms
Ensure: best solution found;
1: Initialize every x^i for the harmony memory, where $i \in \{1, 2, \dots, hms\}$
2: $best \leftarrow$ the best harmony in harmony memory
3: **while** $m > 0$ **do**
 //Memory consideration
4: Call Algorithm 3 to construct a solution x^{new}
 //Pitch adjustment
5: Call Algorithm 4 to enhance the solution x^{new}
 //Update the harmony memory
6: **if** x^{new} is better than the worst solution in the harmony memory **then**
7: Remove the worst solution in the harmony memory
8: Append x^{new} into the harmony memory
9: **end if**
10: **if** x^{new} is better than $best$ **then**
11: $best = x^{new}$
12: **end if**
13: $m = m - 1$
14: **end while**
15: **return** bx

As shown in Algorithm 2, parameters m and hms represent the maximum generation of the SDHS algorithm and the size of the harmony memory. The initial solutions in the harmony memory are randomly created. The initialization of the harmony memory requires the random initialization of n items of the heap hms of harmonies, and its time complexity is $O(hms \times n)$. In line 4, the SDHS algorithm calls the memory consideration operator to construct a new solution x^{new} . In line 5, the SDHS algorithm calls the

solution-level pitch adjustment operator to improve the solution x^{new} . Lines 6 to 9 are used to update the harmony memory. We can use a heap to implement the harmony memory, so the time complexity to maintain the harmony memory is $O(\log_2 hms)$. Lines 10 to 12 are used to record the best solution. Because of the time complexity of both Algorithms 3 and 4 is $O(n)$, the time complexity to construct a solution and maintain the harmony memory is $O(n + \log_2 hms)$. In Algorithm 2, m solutions are constructed in total, so the time complexity of the SDHS algorithm is $O(hms \times n + m \times (n + \log_2 hms))$.

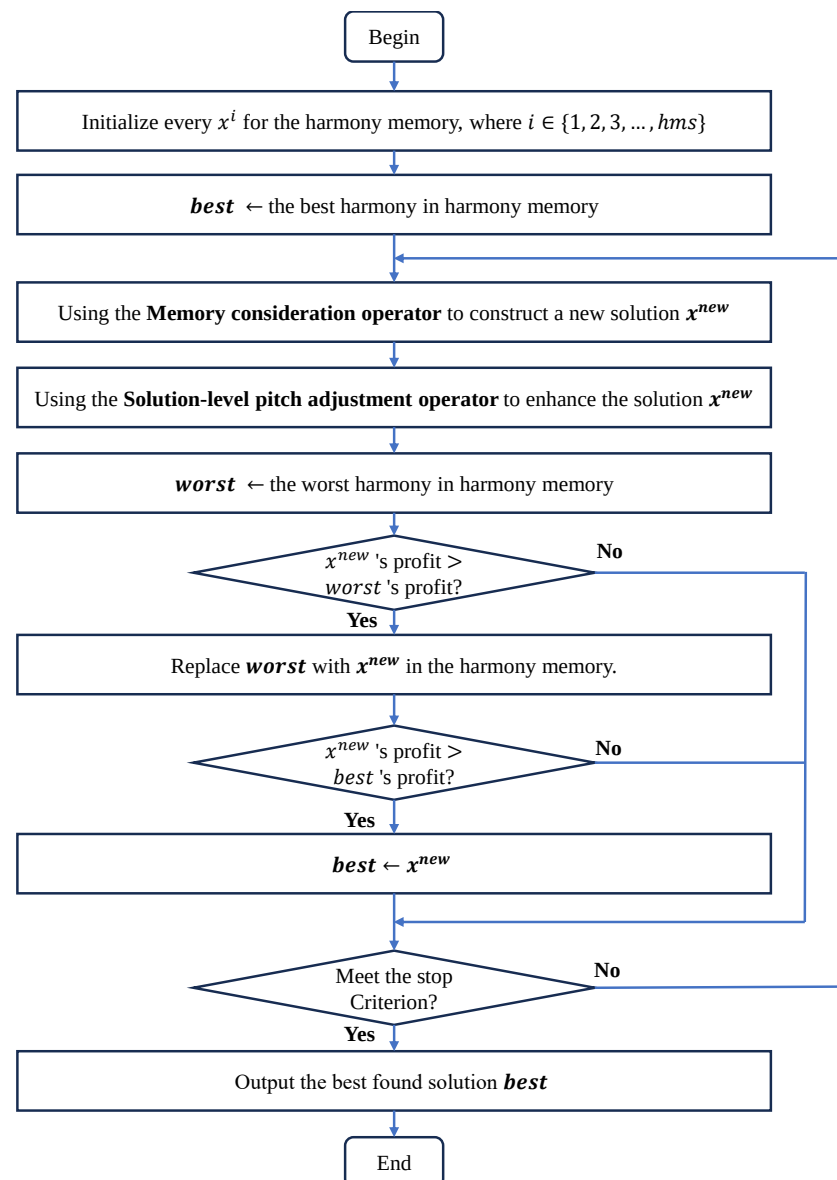


Figure 1. Flowchart of the SDHS algorithm.

As illustrated in Table 1, a comparison is made between the time complexity and parameter settings of basic HS and SDHS. It is evident that both algorithms exhibit analogous initialization and memory consideration complexities. The adoption of a heap structure by SDHS results in a reduction in the complexity of the harmony memory update process from $O(hms)$ to $O(\log_2 hms)$. The algorithm's enhanced adaptability and usability is attributable to the simple, flexible, and efficient operator design, which retains only a single parameter (hms).

To facilitate the comparative study of metaheuristics for the 0-1KP, the source code of the SDHS algorithm has been made publicly available in GitHub (<https://github.com/1497135676/SDHS4KP> (accessed on 8 May 2025)).

Table 1. Time complexity and parameter comparison between HS and SDHS.

Operator	Algorithm	Time Complexity	Parameters	Num Params
Initialization	HS	$O(hms \times n)$	Size of the harmony memory (hms)	1
	SDHS	$O(hms \times n)$	Size of the harmony memory (hms)	1
Memory Consideration	HS	$O(n)$	Harmony Memory Consideration Rate ($hmcr$)	1
	SDHS	$O(n)$	None	0
Pitch Adjustment	HS	$O(n)$	Pitch Adjustment Rate (par), Bandwidth (bw)	2
	SDHS	$O(n)$	None	0
Random Selection	HS	$O(n)$	None	0
	SDHS	None	None	0
Update Harmony Memory	HS	$O(hms)$	None	0
	SDHS	$O(\log_2 hms)$	None	0
Overall per iteration	HS	$O(n + hms)$		
	SDHS	$O(n + \log_2 hms)$		

3.3. Memory Consideration Operator

Algorithm 3 is the memory consideration operator which is used to construct a feasible solution from scratch. In Algorithm 3, items are processed in non-ascending order of the profit–weight ratio under the constriction of the capacity constraint. This heuristic-guided strategy has been widely used by other metaheuristics, such as [13,21,23,24,44,45], to design the greedy repair operator to repair an infeasible solution.

Algorithm 3 The memory consideration operator.

```

1:  $x_j^{new} = 0$ , for  $j = 1, 2, \dots, n$ 
2:  $v = 0$ 
3:  $c = 0$ 
4: for  $k = 1$  to  $n$  do
5:    $j = h_k$ 
6:   if  $c + w_j \leq W$  then
7:     Randomly select a harmony  $x^i$  from the harmony memory
8:      $x_j^{new} = x_j^i$ 
9:     if  $x_j^{new} == 1$  then
10:       $v = v + p_j$ 
11:       $c = c + w_j$ 
12:     end if
13:   end if
14: end for

```

In Algorithm 3, variable x^{new} is a vector which represents the new constructed solution. Each element x_j^{new} of x^{new} indicates whether item j is in the knapsack. In the first line of Algorithm 3, each element of x^{new} is set to 0, which means no item is selected in the beginning. Variable v and variable c represent the total profit and the total weight of the items in the knapsack, respectively. Variable h is a vector which stores the items in non-ascending order of the profit–weight ratio. From line 4 to line 14 of Algorithm 3, items are processed in non-ascending order of the profit–weight ratio. If the selection of an item j does not violate the capacity constraint, then a harmony x^i is randomly selected from the harmony memory and the value of x_j^i is assigned to x_j^{new} . If the value of x_j^{new} is equal to 1, which means item j is selected, then the profit and weight of item j is added to v and

c , respectively. Apparently, the constructed solution is a feasible solution, and the time complexity of Algorithm 3 is $O(n)$. Figure 2 is the corresponding flowchart.

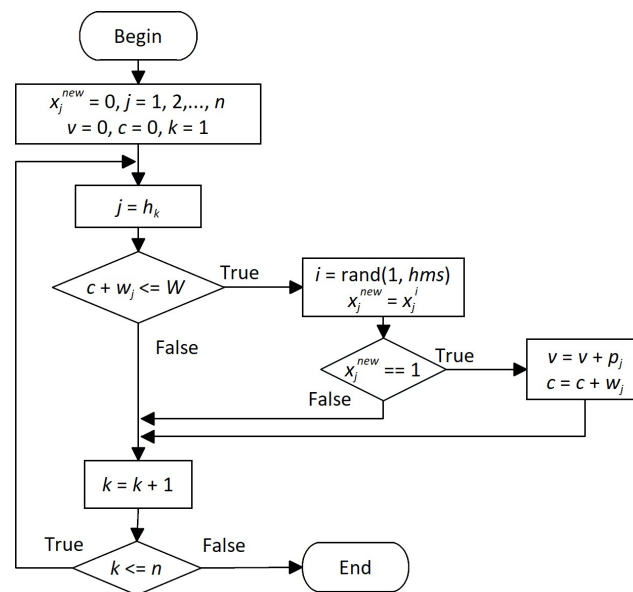


Figure 2. Flowchart of the memory consideration operator.

3.4. Solution-Level Pitch Adjustment Operator

In the field of 0-1KP, the greedy optimization method has been widely used to enhance the feasible solution. Most of the existing metaheuristics, such as [13,21,23,24,44,45], use only the profit–weight ratio to guide the greedy optimization method. Our studies in using trajectory-based metaheuristics for the 0-1KP [47,48] have found that profit information can provide an extra intensification ability for the greedy optimization method. In the SDHS algorithm, we use a profit-guided greedy optimization method, where only profit is used to rank the items. The profit-guided greedy optimization method can be regarded as a solution-level pitch adjustment operator which is used to replace the classical dimension-level pitch adjustment operator of the HS algorithm. Algorithm 4 is the pseudo code of the solution-level pitch adjustment operator.

Algorithm 4 Solution-level pitch adjustment operator.

Require: x, v, c // A feasible solution

- 1: $h \leftarrow$ the sorted items in non-ascending order of profit
 - 2: **for** $k = 1$ to n **do**
 - 3: $j = h_k$
 - 4: **if** $x_j == 0$ and $c + w_j \leq W$ **then**
 - 5: $x_j = 1$
 - 6: $v = v + p_j$
 - 7: $c = c + w_j$
 - 8: **end if**
 - 9: **end for**
 - 10: **return** x, v, c
-

In Algorithm 4, parameters x , v , and c represent a feasible solution, the total profit of the selected items in solution x , and the total weight of the selected items in solution x , respectively. Variable h is a vector which stores the items in non-ascending order of profit. The solution-level pitch adjustment operator examines each item j in non-ascending order of profit and changes x_j from zero to one if the selection of item j does not violate the capacity constraint. Because each item must be checked for whether it can be put into the knapsack to enhance the solution, the time complexity of Algorithm 4 is $O(n)$.

4. Behavior Analysis

In the SDHS algorithm, parameter m controls the maximum number of solutions constructed. Changing this parameter leads to different trade-offs between the solution quality and computation resource used. In the LBSA algorithm [47] and the NM algorithm [48], the maximum number of neighbor solutions is set to 40,000. Therefore, for a fair experimental analysis, the maximum number of solutions constructed by the SDHS algorithm is also set to 40,000.

Except parameter m , there is only one parameter (hms) needed to be fine-tuned for the SDHS algorithm. The three types of 0-1KP instances used by [29] are used in this paper to find a suitable parameter hms and analyze the behaviors of the SDHS algorithm. The features of the three types of instances are listed in Table 2.

Table 2. Features of the three types of 0-1KP instances.

Correlation	Weight w_i	Value v_i	Capacity W
Uncorrelated	rand (10, 100)	rand (10, 100)	$0.75 \times$ sum of weights
Weakly correlated	rand (10, 100)	rand ($w_i - 10, w_i + 10$)	$0.75 \times$ sum of weights
Strongly correlated	rand (10, 100)	$w_i + 10$	$0.75 \times$ sum of weights

Each type contains five 0-1 KP instances, and the numbers of items are 800, 1000, 1200, 1500, and 2000, respectively. These fifteen instances are called KP01, KP02, ..., KP15, respectively. Among those fifteen instances, KP01 to KP05 are uncorrelated instances, KP06 to KP10 are weakly correlated instances, and KP11 to KP15 are strongly correlated instances. The six instances used in this F are KP01, KP04, KP08, KP09, KP14, and KP15. The principle of selecting these six instances is based on our former studies on the LBSA algorithm [47] and the NM algorithm [48]. We want to use difficult instances to tune the parameters of the SDHS algorithm. For uncorrelated instances, KP01 and KP04 are the two instances where the probability to find the best-known solution is less than 50%. For weakly correlated instances, KP08 and KP09 are the two instances where the probability to find the best-known solution is less than 50%. For strongly correlated instances, because both the LBSA algorithm and the NM algorithm can always find the best-known solutions for all instances, we select the two largest instances.

4.1. Effect of Heuristics and Parameter on Performance

The SDHS algorithm has the following three characteristics: (1) there is only one intrinsic parameter, i.e., harmony memory size hms ; (2) the processing order of items in the memory consideration operator is guided by the profit-weight ratio of the item; and (3) the solution-level pitch adjustment operator is guided by the profit of the item. To observe the effect of parameter hms , the contribution of using the profit-weight ratio in the memory consideration operator, and the contribution of using the profit in the pitch adjustment operator, we compare the SDHS algorithm with two variants of the SDHS algorithm, i.e., the SDHS1 algorithm and the SDHS2 algorithm. In the SDHS1 algorithm, the processing order of items is not guided by the profit-weight ratio, which means the items are processed from the first item to the last item in sequence, like in the classical HS algorithm. The comparison between the SDHS algorithm and the SDHS1 algorithm can tell us whether there is positive effect to using the profit-weight ratio to guide the processing order of items. In the SDHS2 algorithm, the solution-level pitch adjustment operator is guided by the profit-weight ratio like the greedy optimization operator used in most metaheuristics for the 0-1KP. The comparison between the SDHS algorithm and the SDHS2 algorithm can tell us whether using the profit to guide the solution-level pitch operator is superior to using the profit-weight ratio.

To find a suitable hms for the SDHS algorithm and to compare the performance of the three SDHS algorithms, we test twenty different values for hms , from 100 to 2000 with a step 100. For each hms , we run the three SDHS algorithms 200 times on each instance and calculate the absolute error of the average profit. Table 3 lists the simulation results of the three SDHS algorithms on the two uncorrelated instances, Table 4 lists the simulation results on the two weakly correlated instances, and Table 5 lists the simulation results on the two strongly correlated instances.

Table 3. Performance comparison among the SDHS algorithm, the SDHS1 algorithm, and the SDHS2 algorithm on KP01 and KP04 instances with different parameter hms .

hms	KP01			KP04		
	SDHS	SDHS1	SDHS2	SDHS	SDHS1	SDHS2
100	0.335	54.28	0.72	0.23	72.95	0.13
200	0.065	44.64	0.31	0.02	60.09	0.025
300	0.01	40.395	0.245	0	53.58	0
400	0.005	38.29	0.145	0	48.3	0
500	0	36.015	0.135	0	47.19	0
600	0	34.775	0.04	0	49.395	0
700	0	33.785	0.05	0	58.07	0
800	0	36.195	0.05	0	71.645	0
900	0	40.72	0.035	0	85.055	0
1000	0	47.105	0.02	0	94.865	0
1100	0	53.61	0.02	0	101.725	0
1200	0	61.65	0.005	0	109.82	0
1300	0	68.865	0.015	0	118.995	0
1400	0	75.695	0.01	0	127.94	0
1500	0	81.925	0.005	0	137.22	0
1600	0	87.77	0.005	0	148.42	0
1700	0	93.475	0	0	158.79	0
1800	0	99.535	0	0	172.145	0
1900	0	105.115	0	0	186.245	0
2000	0	111.095	0	0	199.61	0
Average	0.021	62.247	0.091	0.013	105.103	0.008

Table 3 shows that the SDHS algorithm can always obtain the best-known solution on KP01 and KP04 when the hms is in the range of 500 to 2000 and in the range of 300 to 2000, respectively. Table 4 shows that the SDHS algorithm can always obtain the best-known solution on KP08 and KP09 when the hms is in the range of 300 to 2000 and in the range of 500 to 1200, respectively. Table 5 shows that the SDHS algorithm can always obtain the best-known solution on KP14 and KP15 when the hms is in the range of 100 to 900 and in the range of 100 to 700, respectively. The simulation results of the SDHS algorithm on these six instances show that (1) for each instance, the SDHS algorithm can always obtain the best-known solution in a wide range of hms , and (2) when parameter hms is set to 500, 600, or 700, the SDHS algorithm can always find the best-known solutions on all these six instances.

The simulation results in Tables 3–5 show that the SDHS1 algorithm can never obtain the best-known solution on any of the six instances. Apparently, the performance of the SDHS algorithm is far better than that of the SDHS1 algorithm. Because the only difference between the SDHS algorithm and the SDHS1 algorithm is the processing order of the items, it means that the profit–weight ratio-guided processing order plays an important role in the SDHS algorithm performing well.

Table 3 shows that the SDHS2 algorithm can always obtain the best-known solution on KP01 and KP04 when the *hms* is in the range of 1700 to 2000 and in the range of 300 to 2000, respectively. Table 4 shows that the SDHS2 algorithm can always obtain the best-known solution on KP08 and KP09 when the *hms* is in the range of 600 to 2000 and in the range of 400 to 2000, respectively. Table 5 shows that the SDHS2 algorithm can always obtain the best-known solution on KP14 when the *hms* is in the range of 400 to 1100. The simulation results of the SDHS2 algorithm on these six instances show that (1) there exists a range of parameter *hms* for the SDHS2 algorithm to always obtain the best-known solution on KP01, KP04, KP08, KP09, and KP14, respectively; (2) using the parameters in range of 100 to 2000, the SDHS2 algorithm can never obtain the best solution on KP15; and (3) even on the five instances except KP15, there is not a common *hms* for the SDHS2 algorithm to obtain the best-known solutions on all these five instances. These results indicate that the performance of the SDHS2 algorithm is not as good as that of the SDHS algorithm. Because the only difference between the SDHS algorithm and the SDHS2 algorithm is the heuristic used to guide the processing order of the items in the solution-based pitch adjustment operator, it means that, for the solution-based pitch adjustment operator, the profit-guided processing order is superior to the profit-weight ratio-guided processing order.

Table 4. Performance comparison among the SDHS algorithm, the SDHS1 algorithm, and the SDHS2 algorithm on KP08 and KP09 instances with different parameter *hms*.

<i>hms</i>	KP08			KP09		
	SDHS	SDHS1	SDHS2	SDHS	SDHS1	SDHS2
100	0.18	45.335	0.26	0.385	144.19	0.28
200	0.01	36.445	0.035	0.155	132.495	0.045
300	0	32.74	0.01	0.07	130.785	0.01
400	0	29.99	0	0.01	131.775	0
500	0	28.76	0.005	0	133.07	0
600	0	29.47	0	0	138.56	0
700	0	34.585	0	0	146.825	0
800	0	44.16	0	0	157.915	0
900	0	55.18	0	0	169.455	0
1000	0	65.18	0	0	182.29	0
1100	0	72.32	0	0	195.82	0
1200	0	79.75	0	0	209.89	0
1300	0	85.14	0	0.005	226.31	0
1400	0	91.58	0	0.02	242.24	0
1500	0	98.17	0	0.425	259.445	0
1600	0	105.375	0	0.905	276.425	0
1700	0	114.675	0	1.285	295.605	0
1800	0	126.37	0	2.55	313.225	0
1900	0	138.885	0	4.265	331.26	0
2000	0	153.075	0	6.78	348.215	0
Average	0.010	73.359	0.016	0.843	208.290	0.017

Zhao et al. [25] analyzed that only using the profit-weight ratio may fail to find the best solution for the construction method in some cases. Similarly, only using the profit-weight ratio may also fail to find best solution for metaheuristics in some cases. For example, suppose the remaining capacity of package is 30 and there are 3 items not in the package. The profits and weights of those three items are [20, 25, 25] and [10, 15, 15], respectively. Then, the profit-weight ratio of those three items are [2, 5/3, 5/3]. If the profit-weight ratio is used to guide the pitch adjustment operator, then the total profit will increase by 45. But if the profit is used to guide the pitch adjustment operator, then the

total profit will increase by 50. In the SDHS algorithm, the memory consideration operator is guided by the profit–weight ratio, and the pitch adjustment operator is guided by the profit. These two heuristics cooperate with each other to guarantee good performance for the SDHS algorithm. To further experimentally analyze whether this result is reliable, we carry out comparative experiment on six types of 0-1KP instances with a wide range of parameters. The six types include uncorrelated (UC), weakly correlated (WC), strongly correlated (SC), multiple strongly correlated (MSC), profit ceiling (PC), and circle (CI) instances. The weights are distributed in $[1, R]$ where R is in 100, 1000, 10,000. The profits are created according to the type of instance [62]. The value of n (number of items) is from 1000 to 2000 with a step of 100. For each combination of type, R , and n , 10 instances are created. In total, 1980 instances ($6 \times 3 \times 11 \times 10$) are used in the comparative experiment. For each instance, we run the methods 50 times and use the mean value to compare the performance of using profit and using profit–weight ratio. Table 6 lists the statistical values of the experimental results, where columns $>$, $=$, and $<$ list the number of instances where using profit has better, equal, or worse performance, respectively. Column p -value lists the results of the Wilcoxon signed ranks test which is highlighted in bold if using profit is significantly better than using profit–weight ratio. In case the two strategies obtain exactly same results on 110 instances, the corresponding cells of p -value are labeled as ‘-’. Among the 18 combination of type and R , using profit is significantly better than using the profit–weight ratio on 12 cases, has no significant difference on 5 cases, and is significantly worse only on 1 case when the type is PC and R is 10,000. These results confirm that using the profit to guide the solution-level pitch adjustment operator is superior to using the profit–weight ratio.

Table 5. Performance comparison among the SDHS algorithm, the SDHS1 algorithm, and the SDHS2 algorithm on KP014 and KP15 instances with different parameter hms .

hms	KP14			KP15		
	SDHS	SDHS1	SDHS2	SDHS	SDHS1	SDHS2
100	0	72.59	0.46	0	106.425	8.285
200	0	85.455	0.01	0	121.04	3.37
300	0	95.335	0.005	0	134.045	2.035
400	0	103.92	0	0	147.27	1.365
500	0	110.86	0	0	155.245	1.115
600	0	117.69	0	0	162.93	1.035
700	0	121.46	0	0	169.12	0.995
800	0	125.785	0	2.015	173.815	0.78
900	0	128.85	0	9.85	177.06	1.22
1000	1.095	130.995	0	10.015	180.01	2.56
1100	8.745	134.57	0	17.455	182.77	5.565
1200	9.955	136.355	0.01	20.03	184.91	9.9
1300	10.665	137.585	0.03	25.835	186.475	10
1400	16.435	137.82	0.11	29.675	187.975	10
1500	19.11	139.77	1.14	34.565	188.55	10
1600	20.47	139.21	6.765	39.56	190.01	10.325
1700	24.65	140.48	9.225	42.86	189.42	13.81
1800	28.39	140.915	9.975	48.145	190.145	17.88
1900	29.91	141.395	10	50.53	189.98	19.53
2000	33.585	142.14	10.005	55.785	190.21	19.775
Average	10.151	124.159	2.387	19.316	170.370	7.477

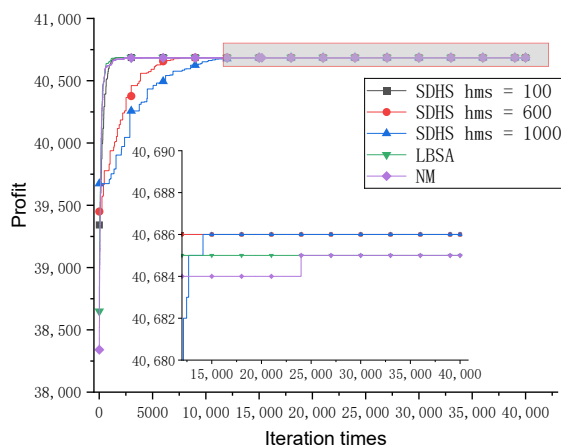
Table 6. Performance comparison between profit and profit–weight ratio for the solution-level pitch adjustment operator.

Type	R = 100				R = 1000				R = 10,000			
	>	=	<	<i>p</i>	>	=	<	<i>p</i>	>	=	<	<i>p</i>
UC	14	94	2	1.5×10^{-3}	48	38	24	1.3×10^{-2}	60	30	20	2.8×10^{-6}
WC	1	109	0	1.0	32	33	45	6.6×10^{-1}	53	9	48	4.1×10^{-1}
SC	66	44	0	1.7×10^{-12}	78	32	0	1.7×10^{-14}	101	9	0	2.7×10^{-18}
MSC	61	49	0	1.1×10^{-11}	72	38	0	1.7×10^{-13}	85	25	0	1.2×10^{-15}
PC	0	110	0	-	0	110	0	-	8	69	33	5.0×10^{-4}
CI	105	0	5	3.6×10^{-19}	110	0	0	8.9×10^{-20}	110	0	0	8.9×10^{-20}

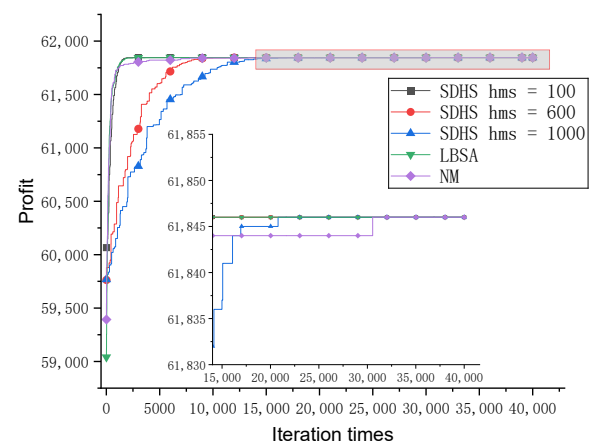
Note: Bold values in the *p* columns indicate statistical significance ($p < 0.05$).

4.2. Effect of Parameter *hms* on the Convergence Process

To analyze the effect of parameter *hms* on the convergence process of the SDHS algorithm, we compare the convergence process of the SDHS algorithms with three different values of parameter *hms*, i.e., 100, 600, and 1000. Figure 3a–f show the convergence process of the SDHS algorithm on KP01, KP04, KP08, KP09, KP14, and KP15, respectively. Those figures clearly show that parameter *hms* controls the convergence speed of the SDHS algorithm. If parameter *hms* is too small, the selection pressure will be too strong, and then the convergence speed will be too quick, and it may lead to premature convergence easily. On the contrary, if parameter *hms* is too large, the selection pressure will be too weak, and then the convergence speed will be too slow, and it may lead to insufficient exploitation. A suitable parameter *hms* is important for the SDHS algorithm to have good convergence behavior.

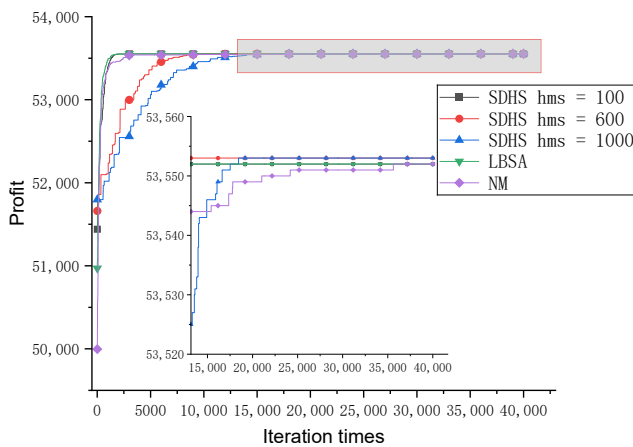


(a) Convergence process of profit on KP01 instance

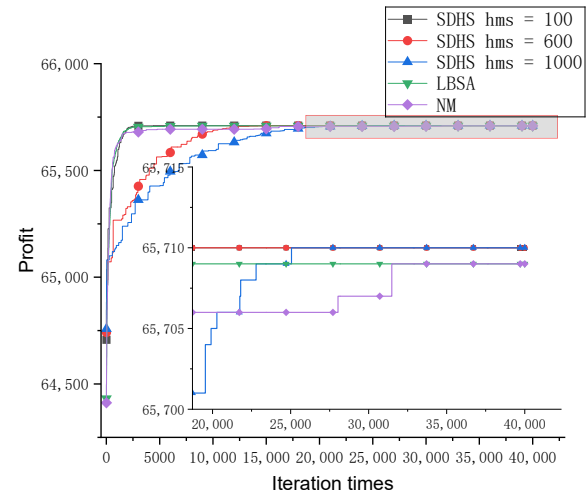


(b) Convergence process of profit on KP03 instance

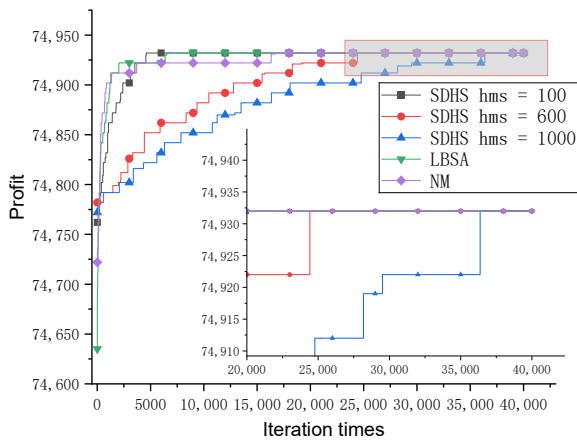
Figure 3. Cont.



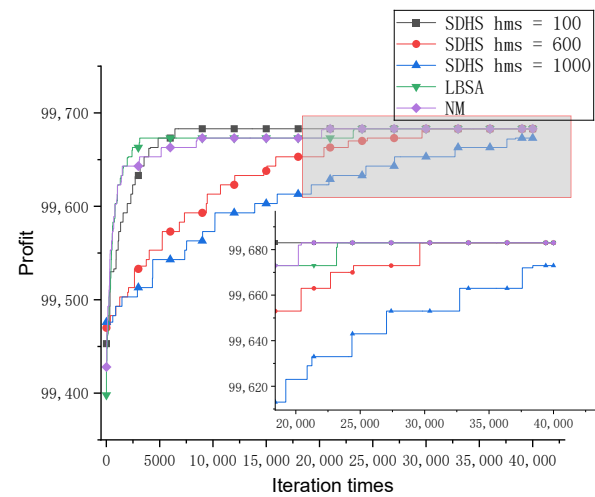
(c) Convergence process of profit on KP08 instance



(d) Convergence process of profit on KP09 instance



(e) Convergence process of profit on KP14 instance



(f) Convergence process of profit on KP15 instance

Figure 3. On the six examples KP01, KP03, KP08, KP09, KP14, and KP15, the SDHS profit convergence curves at $hms = 100, 600, 1000$, as well as the convergence processes of LBSA and NM.

4.3. Effect of Parameter hms on the Diversity

To analyze the effect of parameter hms on the diversity of the SDHS algorithm, we use the entropy of the solutions in harmony memory to measure the diversity of the SDHS algorithm, and compare the entropy of the SDHS algorithms with three different values of parameter hms , i.e., 100, 600, and 1000. We use $f_{j,1}$ to represent the frequency when x_j is equal to 1, and use $f_{j,0}$ to represent the frequency when x_j is equal to 0. Then, the entropy for the selection of item j can be defined as

$$H_j = - \sum_{i=0}^1 f_{j,i} \log_2(f_{j,i}) \quad (16)$$

The maximum of Equation (16) occurs when the selecting and discarding of the item j are equally likely. The minimum of Equation (16) occurs when all solutions select item j or no solution selects item j . That means a value of 1 represents the largest diversity for the

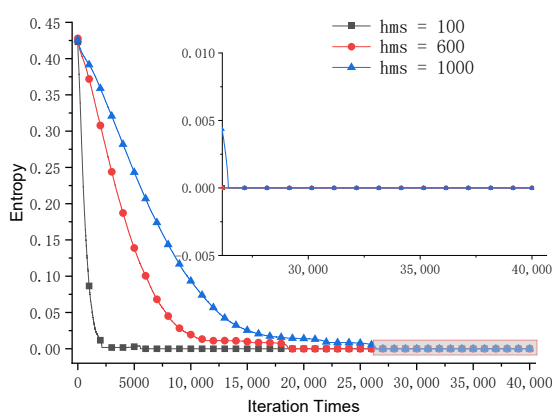
selection of item j , and 0, no diversity. The entropy of the solutions in harmony memory is defined as the average of the H_j :

$$H = \frac{\sum_{j=1}^n H_j}{n} \quad (17)$$

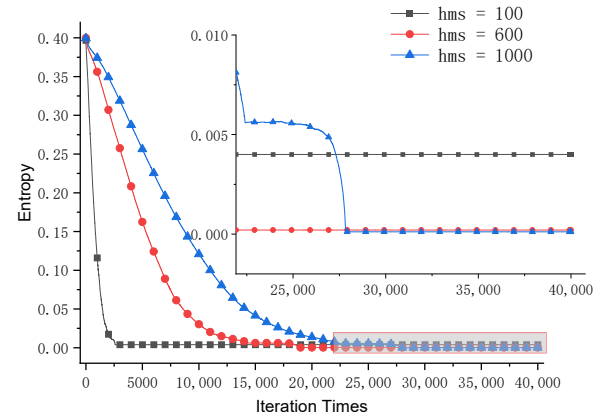
where n is the number of items. Apparently, the value of H is between 0 and 1. When the H is equal to 1, the SDHS algorithm has the largest diversity. When H is equal to 0, the SDHS algorithm does not have any diversity and we cannot construct a new solution after that.

Figure 4a–f are the entropy of the SDHS algorithm on KP01, KP04, KP08, KP09, KP14, and KP15, respectively.

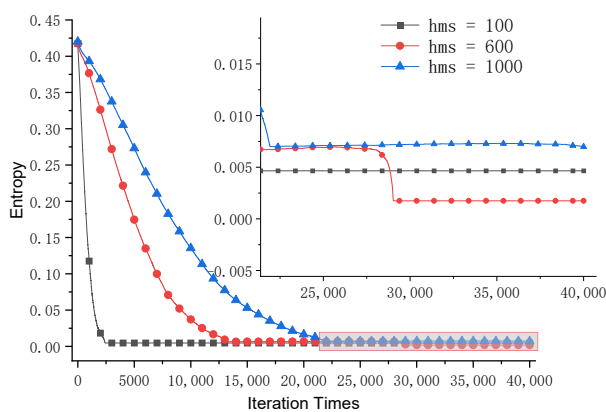
These figures clearly show that parameter hms controls the decrease speed of the entropy for the SDHS algorithm. As the entropy is used to measure the diversity, if parameter hms is too small, the decrease speed of diversity will be too quick and it may lead to premature convergence easily. On the contrary, if parameter hms is too large, the decrease speed of diversity will be too slow and it may lead to insufficient exploitation. A suitable parameter hms is important for the SDHS algorithm to have enough diversity.



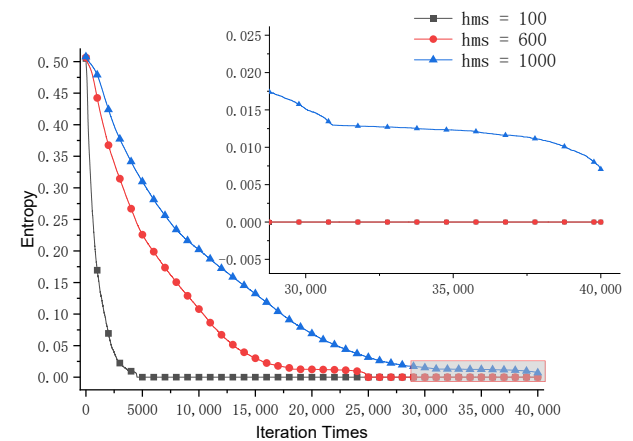
(a) Convergence process of profit on KP01 instance



(b) Convergence process of profit on KP03 instance

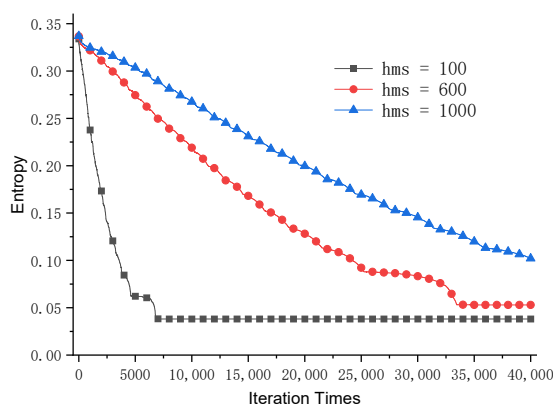


(c) Convergence process of profit on KP08 instance

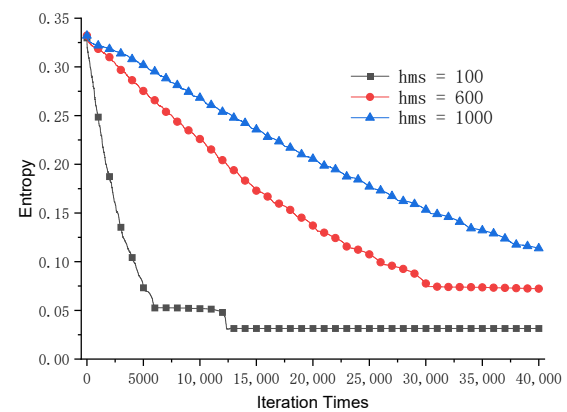


(d) Convergence process of profit on KP09 instance

Figure 4. Cont.



(e) Convergence process of profit on KP14 instance



(f) Convergence process of profit on KP15 instance

Figure 4. Entropy convergence process of the SDHS algorithm on KP01, KP04, KP08, KP09, KP14, and KP15.

4.4. The Running Time of SDHS

The significance of using the metaheuristic for the 0-1KP is that the time complexity and space complexity of the metaheuristic is independent of the capacity of package. As reported in [62], dynamic programming (DP) algorithms fail when the capacity of package is large, and branch and bound methods fail in difficult instances even with a small number of items, such as multiple strongly correlated, profit ceiling, and circle instances. There are numerous instances for which all currently known exact methods perform badly [62]. Using DP as an example, both its time complexity and space complexity are $O(nW)$, which means DP may run out of space if the capacity of package is large enough. Even if the memory is enough, due to the cost of managing large memory, the running time of DP may increase quickly if the capacity of package is large. We use a set of uncorrelated instances to compare the running time of SDHS and DP. The item number is set to 1000, and the weights are distributed in $[1, R]$ where R begins from 1000 and increases linearly with a step 500. In our experiment, the maximum memory of JVM is set to 5G. Our experiment finds that when R is 3500, the DP fails due to running out of memory. Figure 5 compares the running time of SDHS and DP on instances with different parameter R . Figure 5 clearly shows that the running time of SDHS is independent of the capacity of package, but the running time of DP increases sharply when the capacity of package is large.

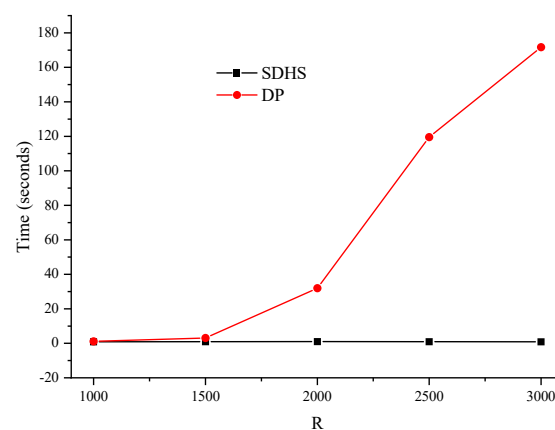


Figure 5. Compare the running time of SDHS and DP.

5. Comparative Experiments

To observe the competitiveness of the SDHS algorithm, the performance of the SDHS algorithm is compared with thirteen state-of-the-art algorithms on three sets of large-scale 0-1KP instances. These three instances have been widely used to evaluate the performance of 0-1 knapsack problem algorithms. Table 7 shows the main information for these three example groups. The performance test of SDHS on the *hms* parameter is discussed in detail in Section 3. We set *hms* = 600 as the setting for subsequent experiments, which can make SDHS maintain good diversity and convergence speed.

The first set of large-scale 0-1KP instances, which includes sixteen instances, was proposed by Kong et al. in [45]. The numbers of items of these sixteen instances are 100, 200, 300, 500, 700, 1000, 1200, 1500, 1800, 2000, 2600, 3000, 3500, 4900, 5800, and 6400, respectively. These sixteen instances are named LKP01–LKP16. For each instance, the profit of each item is randomly produced from [0.5, 1], its weight is randomly produced from [0.5, 2], and the capacity of the knapsack is equal to 0.75 times of the total weights of the items. Apparently, these sixteen instances belong to uncorrelated instances. The second set of large-scale 0-1KP instance includes three types of 0-1KP instances, i.e., uncorrelated instances, weakly correlated instances, and strongly correlated instances. Each type contains five 0-1KP instances, and the numbers of items are 800, 1000, 1200, 1500, and 2000, respectively. The third set of large-scale 0-1KP instance includes five types of 0-1KP instances, i.e., weakly correlated instances [51], strongly correlated instances [51], multiple strongly correlated instances [51], profit ceiling instances [51], and uncorrelated instances [45]. Each of the first two types contains five instances, and the numbers of items are 200, 300, 500, 800, and 1000, respectively. Each of the middle two types contains five instances, and the numbers of items are 300, 500, 800, 1000, and 1200, respectively. The last type contains four instances, and the numbers of items are 3000, 5000, 7000, and 10,000, respectively.

In Section 5.1, we use the first set of large-scale 0-1KP instances to compare the SDHS algorithm with three HS algorithms. In Section 5.2, we use the second set of large-scale 0-1KP instances to compare the SDHS algorithm with six state-of-the-art metaheuristics. In Section 5.3, we use the third set of large-scale 0-1KP instances to compare the SDHS algorithm with four state-of-the-art metaheuristics. The SDHS algorithm is coded with Java and conducted on an Intel Core i9-13900HX CPU (2.4 GHz), 16 GB of memory, and Windows 11.

Table 7. The three large-scale 0-1KP examples selected for this experiment.

0-1KP Instances	Data Set Classification	Item Number	Algorithms
The first instance set	16 uncorrelated instances	500–6400	NGHS [39], ABHS [41], SBHS [45]
The second instance set	5 uncorrelated instances	800–2000	CSGHS [9], BMBO [27],
	5 weakly correlated instances	800–2000	CMBO [29], OMBO [28],
	5 strongly correlated instances	800–2000	NM [48], LBSA [47]
The third instance set	5 weakly correlated instances	200–1000	CGMA [13], MBPSO [30],
	5 strongly correlated instances	200–1000	DGHS [44], HSOSHS [21]
	5 multiple strongly correlated instances	300–1200	
	5 profit ceiling instances	300–1200	
	4 uncorrelated instances	3000–10,000	

5.1. Experiment on the First Set of Large-Scale 0-1KP Instances

Using the first set of large-scale 0-1KP instances, we compare the SDHS algorithm with three HS algorithms, the novel global harmony search (NGHS) algorithm [39], improved adaptive binary harmony search (ABHS) algorithm [41], and simplified binary harmony

search (SBHS) algorithm [45]. In [45], the maximum numbers of iterations for these instances are set to 15,000, 15,000, 20,000, 20,000, 30,000, 30,000, 40,000, 40,000, 50,000, 50,000, 60,000, 80,000, 100,000, 120,000, 150,000, and 200,000. In the SDHS algorithm, the number of iterations is set to the value used by [45] or 40,000 if the value used by [45] is bigger than 40,000. It means that the SDHS algorithm only constructs 20 percent of the number of solutions constructed by its competitors on the largest instance.

As in [45], the SDHS algorithm is run 30 times for each instance. The statistical results of 30 independent runs are reported in Table 8. The best results are highlighted in bold. The data of [39,41,45] are taken from [45]. In terms of the worst value and mean value, the SDHS algorithm obtains better results than the other three algorithms on all sixteen instances. In terms of the best value, the SDHS algorithm finds a new best solution on nine instances, i.e., LKP07, LKP08, LKP09, LKP11, LKP12, LKP13, LKP14, LKP15, and LKP16. These results clearly show that the SDHS algorithm has best performance among the four HS algorithms.

Table 8. Comparison of the SDHS algorithm with three HS algorithms.

Method	Ins	Best	Worst	Mean	Time (s)	Ins	Best	Worst	Mean	Time (s)
NGHS	LKP01	61.82	61.11	61.5	-	LKP09	1133.44	1125.69	1129.02	-
ABHS		62.01	61.71	61.9	-		1140.69	1133.22	1136.57	-
SBHS		62.08	61.97	62.04	-		1155.65	1155.35	1155.57	-
SDHS		62.08	62.08	62.08	0.02		1155.68	1155.68	1155.68	1.04
NGHS	LKP02	128.34	126.87	127.66	-	LKP10	1257.45	1249.74	1252.86	-
ABHS		129.31	128.51	128.94	-		1263.67	1257.85	1260.46	-
SBHS		129.44	129.27	129.37	-		1283.92	1283.26	1283.79	-
SDHS		129.44	129.44	129.44	0.04		1283.92	1283.92	1283.92	1.16
NGHS	LKP03	190.18	187.9	189.23	-	LKP11	1615.64	1604.28	1610.5	-
ABHS		191.49	190.32	191.04	-		1623.3	1613.54	1618.77	-
SBHS		192.02	191.85	192.01	-		1653.72	1653.43	1653.64	-
SDHS		192.02	192.02	192.02	0.07		1653.76	1653.76	1653.76	1.64
NGHS	LKP04	310.16	305.67	308.33	-	LKP12	1877.6	1868.31	1872.43	-
ABHS		312.51	310.67	311.79	-		1879.12	1868.61	1874.04	-
SBHS		314.23	314.1	314.19	-		1917.49	1917.23	1917.42	-
SDHS		314.23	314.23	314.23	0.12		1917.58	1917.57	1917.57	1.96
NGHS	LKP05	442.32	436.45	440.83	-	LKP13	2200.57	2191.68	2196.15	-
ABHS		446.3	444.42	445.43	-		2203.56	2193.78	2199.31	-
SBHS		448.65	448.46	448.6	-		2248.27	2247.77	2248.12	-
SDHS		448.65	448.65	448.65	0.27		2248.30	2248.30	2248.30	2.42
NGHS	LKP06	626.77	619.15	623.87	-	LKP14	3061.12	3047.8	3054.33	-
ABHS		632.38	628.65	630.34	-		3055.47	3040.32	3046.91	-
SBHS		638.14	638	638.09	-		3135.71	3135.29	3135.58	-
SDHS		638.14	638.14	638.14	0.45		3135.77	3135.77	3135.77	3.83
NGHS	LKP07	750.67	745.05	747.66	-	LKP15	3625.86	3609.96	3617.04	-
ABHS		756.08	752.10	754.26	-		3615.85	3601.58	3607.61	-
SBHS		763.81	763.39	763.71	-		3707.39	3706.98	3707.29	-
SDHS		763.83	763.83	763.83	0.68		3707.45	3707.44	3707.44	4.99
NGHS	LKP08	945.20	938.31	941.97	-	LKP16	4009.33	3995.04	4003.40	-
ABHS		950.70	947.36	949.17	-		4009.08	3993.47	4001.41	-
SBHS		964.91	964.70	964.85	-		4090.83	4090.36	4090.64	-
SDHS		964.92	964.91	964.92	0.97		4090.87	4090.86	4090.87	5.64

Note: Bold numbers indicate the algorithm found the relatively best solution.

In addition, the comparison results between SDHS and the average values of the competitive algorithms based on the Friedman test and Holm post hoc test are shown in Table 9. As shown in Table 9, according to the Friedman test, SDHS obtains the best

ranking. Specifically, compared with NGHS and ABHS, SDHS has a significant advantage. Although the performance difference is not significant compared with the SBHS algorithm, SDHS obtains a better ranking, and the average values are all better than those of other algorithms. The table also counts the percentage of improvement of SDHS compared with the reported approach, calculated as Equation (18):

$$\text{Gap}(A) = \frac{\text{Mean}_A - \text{Mean}_{\text{SDHS}}}{\text{Mean}_A} \quad (18)$$

Table 9. The performance summary between SDHS, NGHS, ABHS, and SBHS; the average ranks are computed by the Friedman test and the significance is assessed by Holm’s post hoc method.

Algorithm	Rank	Gap (%)	<i>p</i> -Value	<i>p</i> -Holm	Sum of +/=/-
SDHS	1	-	-	-	-
NGHS	3.81	2.128	4.31×10^{-9}	1.29×10^{-8}	0/0/16
ABHS	3.19	1.565	9.80×10^{-6}	1.96×10^{-5}	0/0/16
SBHS	2	0.015	0.126	0.126	0/0/16

Note: Bold values in the *p* columns indicate statistical significance ($p < 0.05$).

5.2. Experiment on the Second Set of Large-Scale 0-1KP Instances

Using the fifteen large-scale 0-1KP instances introduced in Section 4, the SDHS algorithm is compared with six state-of-the-art metaheuristics, i.e., the hybrid cuckoo search algorithm with global harmony search (CSGHS) [9], binary monarch butterfly optimization (BMBO) algorithm [27], chaotic monarch butterfly optimization (CMBO) algorithm [29], opposition-based learning monarch butterfly optimization (OMBO) algorithm [28], list-based SA (LBSA) algorithm [47], and nosing method (NM) [48]. As in the LBSA algorithm and the NM, the maximum number of iterations is set to 40,000. The comparisons are listed in Tables 10–12, where the best results are highlighted in bold. Because the CSGHS algorithm and the BMBO algorithm are not tested on instances with 2000 items, the corresponding cells of those two algorithms are labeled as ‘-’.

Table 10 shows that the CSGHS algorithm and the BMBO algorithm cannot find the best-known solution on any instance. Both the CMBO algorithm and the OMBO algorithm cannot find the best-known solution on the KP03 instance. The NM cannot find the best-known solution on the KP01 instance. Both the LBSA algorithm and the SDHS algorithm can find the best-known solutions on all uncorrelated instances, but only the SDHS algorithm can always find best-known solutions. Table 11 shows that the CSGHS algorithm and the BMBO algorithm cannot find the best-known solution on any instance. The CMBO algorithm cannot find the best-known solution on the KP08 instance. Both the LBSA algorithm and the NM cannot find the best-known solution on the KP09 instance. Both the OMBO algorithm and the SDHS algorithm can find the best-known solutions on all weakly correlated instances, but only the SDHS algorithm can always find best-known solutions. Table 12 shows that the CSGHS algorithm and the BMBO algorithm cannot find the best-known solution on any instance. The CMBO algorithm cannot find the best-known solution on the KP12 instance. All the other four algorithms can find the best-known solutions on all strongly correlated instances, but only the NM, the LBSA algorithm and the SDHS algorithm can always find best-known solutions. In summary, only the SDHS algorithm can always find the best-known solutions on all 15 instances.

Table 10. Comparison of the SDHS algorithm with six metaheuristics on large-scale uncorrelated 0-1KP instances.

Ins	Opt	Method	Best	Worst	Mean	Median	Std	Time (s)
KP01	40,686	CSGHS	40,342	40,056	40,182	40,190	68.87	-
		BMBO	40,232	39,765	40,035	40,036	105.8	-
		CMBO	40,686	40,683	40,683	40,683	0.71	-
		OMBO	40,686	40,683	40,684	40,683	0.86	-
		NM	40,685	40,684	40,684.88	40,685	0.22	-
		LBSA	40,686	40,684	40,684.9	40,685	0.18	-
		SDHS	40,686	40,686	40,686	40,686	0	0.38
KP02	50,592	CSGHS	50,027	49,717	49,846	49,835	84.36	-
		BMBO	50,024	49,336	49,699	49,689	135.3	-
		CMBO	50,592	50,590	50,590	50,590	0.49	-
		OMBO	50,592	50,590	50,590	50,590	0.70	-
		NM	50,592	50,592	50,592	50,592	0	-
		LBSA	50,592	50,591	50,591.98	50,592	0.04	-
		SDHS	50,592	50,592	50,592	50,592	0	0.55
KP03	61,846	CSGHS	60,951	60,616	60,788	60,791	79.79	-
		BMBO	61,109	60,214	60,677	60,660	165.8	-
		CMBO	61,845	61,840	61,841	61,840	1.38	-
		OMBO	61,845	61,840	61,842	61,843	1.82	-
		NM	61,846	61,845	61,845.32	61,845	0.44	-
		LBSA	61,846	61,845	61,845.27	61,845	0.40	-
		SDHS	61,846	61,846	61,846	61,846	0	0.79
KP04	77,033	CSGHS	75,889	75,452	75,639	75,631	112.3	-
		BMBO	75,761	75,062	75,464	75,482	193.3	-
		CMBO	77,033	77,031	77,031	77,031	0.31	-
		OMBO	77,033	77,031	77,031	77,031	0.56	-
		NM	77,033	77,032	77,032.92	77,033	0.15	-
		LBSA	77,033	77,032	77,032.76	77,033	0.37	-
		SDHS	77,033	77,033	77,033	77,033	0	0.9
KP05	102,316	CSGHS	-	-	-	-	-	-
		BMBO	-	-	-	-	-	-
		CMBO	102,316	102,313	102,314	102,313	0.93	-
		OMBO	102,316	102,313	102,314	102,313	1.11	-
		NM	102,316	102,316	102,316	102,316	0	-
		LBSA	102,316	102,315	102,315.9	102,316	0.20	-
		SDHS	102,316	102,316	102,316	102,316	0	1.15

Note: Bold numbers indicate the algorithm found the relatively best solution.

The comparison results between SDHS and the average values of the competitive algorithms, including CSGHS, BMBO, CMBO, OMBO, NM, and LBSA, are detailed in Table 13. These findings are derived from the Friedman test and Holm's post hoc test, which provide a robust statistical framework for evaluating algorithmic performance. According to the Friedman test, SDHS achieves the highest ranking with a value of 1.71, outperforming all other algorithms. Notably, SDHS exhibits a statistically significant advantage over CSGHS and BMBO. In contrast, when compared to CMBO, OMBO, NM, and LBSA, the performance differences are not statistically significant; nevertheless, SDHS consistently secures a superior ranking. The gap values further corroborate the dominance of SDHS: the smallest differences against NM and LBSA are 0.0005, while larger gaps of 0.932 and 1.071 are observed against CSGHS and BMBO, respectively. Additionally, the comparison summary reveals no instances where SDHS underperformed relative to its counterparts as reflected by the (+/ = /-) notation sums (e.g., 0/0/12 for CSGHS and BMBO). Collectively, these results highlight the exceptional and consistent performance of SDHS across all evaluated scenarios, positioning it as a leading algorithm in this comparative analysis.

Table 11. Comparison of the SDHS algorithm with six metaheuristics on large-scale weakly correlated 0-1KP instances.

Ins	Opt	Method	Best	Worst	Mean	Median	Std	Time (s)
KP06	35,069	CSGHS	34,850	34,795	34,824	34,825	14.00	-
		BMBO	34,860	34,681	34,786	34,784	35.01	-
		CMBO	35,069	35,064	35,067	35,067	1.45	-
		OMBO	35,069	35,064	35,067	35,068	1.47	-
		NM	35,069	35,069	35,069	35,069	0	-
		LBSA	35,069	35,068	35,068.99	35,069	0.02	-
		SDHS	35,069	35,069	35,069	35,069	0	0.38
KP07	43,786	CSGHS	43,484	43,386	43,440	43,442	22.39	-
		BMBO	43,491	43,359	43,412	43,413	31.36	-
		CMBO	43,786	43,781	43,784	43,784	1.34	-
		OMBO	43,786	43,782	43,785	43,785	1.03	-
		NM	43,786	43,785	43,785.96	43,786	0.08	-
		LBSA	43,786	43,785	43,785.97	43,786	0.06	-
		SDHS	43,786	43,786	43,786	43,786	0	0.63
KP08	53,553	CSGHS	52,711	52,354	52,556	52,565	76.89	-
		BMBO	52,774	52,110	52,425	52,390	158.2	-
		CMBO	53,552	53,552	53,552	53,552	0	-
		OMBO	53,553	53,552	53,552	53,552	1.82	-
		NM	53,553	53,552	53,552.02	53,552	0.04	-
		LBSA	53,553	53,552	53,552.03	53,552	0.06	-
		SDHS	53,553	53,553	53,553	53,553	0	0.69
KP09	65,710	CSGHS	65,116	64,980	65,045	65,044	38.14	-
		BMBO	65,123	64,916	65,022	65,012	56.38	-
		CMBO	65,710	65,708	65,709	65,708	0.58	-
		OMBO	65,710	65,708	65,709	65,709	0.52	-
		NM	65,709	65,709	65,709	65,709	0	-
		LBSA	65,709	65,709	65,709	65,709	0	-
		SDHS	65,710	65,710	65,710	65,710	0	0.76
KP10	108,200	CSGHS	-	-	-	-	-	-
		BMBO	-	-	-	-	-	-
		CMBO	108,200	108,200	108,200	108,200	0	-
		OMBO	108,200	108,200	108,200	108,200	0	-
		NM	108,200	108,200	108,200	108,200	0	-
		LBSA	108,200	108,200	108,200	108,200	0	-
		SDHS	108,200	108,200	108,200	108,200	0	1.14

Note: Bold numbers indicate the algorithm found the relatively best solution.

Table 12. Comparison of the SDHS algorithm with six metaheuristics on large-scale strongly correlated 0-1KP instances.

Ins	Opt	Method	Best	Worst	Mean	Median	Std	Time (s)
KP11	40,167	CSGHS	40,147	40,126	40,132	40,130	5.54	-
		BMBO	40,127	40,107	40,116	40,117	4.52	-
		CMBO	40,167	40,166	40,167	40,167	0.14	-
		OMBO	40,167	40,167	40,167	40,167	0	-
		NM	40,167	40,167	40,167	40,167	0	-
		LBSA	40,167	40,167	40,167	40,167	0	-
		SDHS	40,167	40,167	40,167	40,167	0	0.65

Table 12. Cont.

Ins	Opt	Method	Best	Worst	Mean	Median	Std	Time (s)
KP12	49,443	CSGHS	49,403	49,383	49,393	49,393	6.52	-
		BMBO	49,393	49,353	49,378	49,382	10.12	-
		CMBO	49,433	49,433	49,422	49,433	2.49	-
		OMBO	49,443	49,441	49,443	49,443	0.34	-
		NM	49,443	49,443	49,443	49,443	0	-
		LBSA	49,443	49,443	49,443	49,443	0	-
		SDHS	49,443	49,443	49,443	49,443	0	0.85
KP13	60,640	CSGHS	60,587	60,567	60,573	60,570	5.32	-
		BMBO	60,588	60,530	60,562	60,560	11.98	-
		CMBO	60,640	60,639	60,640	60,640	0.14	-
		OMBO	60,640	60,640	60,640	60,640	0	-
		NM	60,640	60,640	60,640	60,640	0	-
		LBSA	60,640	60,640	60,640	60,640	0	-
		SDHS	60,640	60,640	60,640	60,640	0	0.74
KP14	74,932	CSGHS	74,858	74,817	74,835	74,832	9.31	-
		BMBO	74,842	74,772	74,818	74,821	15.80	-
		CMBO	74,932	74,931	74,932	74,932	0.27	-
		OMBO	74,932	74,931	74,932	74,932	0.14	-
		NM	74,932	74,932	74,932	74,932	0	-
		LBSA	74,932	74,932	74,932	74,932	0	-
		SDHS	74,932	74,932	74,932	74,932	0	0.88
KP15	99,683	CSGHS	-	-	-	-	-	-
		BMBO	-	-	-	-	-	-
		CMBO	99,683	99,672	99,682	99,683	2.23	-
		OMBO	99,683	99,679	99,683	99,683	0.58	-
		NM	99,683	99,683	99,683	99,683	0	-
		LBSA	99,683	99,683	99,683	99,683	0	-
		SDHS	99,683	99,683	99,683	99,683	0	1.25

Note: Bold numbers indicate the algorithm found the relatively best solution.

Table 13. The performance summary between SDHS, CSGHS, BMBO, CMBO, OMBO, NM, and LBSA; the average ranks are computed by the Friedman test and the significance is assessed by Holm's post hoc method.

Algorithm	Rank	Gap (%)	<i>p</i> -Value	<i>p</i> -Holm	Sum of + / = / -
SDHS	1.71	-	-	-	-
CSGHS	7	0.932	2.35×10^{-5}	1.17×10^{-4}	0/0/12
BMBO	6	1.071	4.13×10^{-8}	2.48×10^{-7}	0/0/12
CMBO	4.21	0.005	0.069	0.275	0/3/9
OMBO	3.75	0.002	0.237	0.71	0/4/8
NM	2.58	0.0005	0.956	1	0/7/5
LBSA	2.75	0.0005	0.901	1	0/7/5

Note: Bold values in the *p* columns indicate statistical significance ($p < 0.05$).

5.3. Experiment on the Third Set of Large-Scale 0-1KP Instances

Using the third set of large-scale 0-1KP instances, the SDHS algorithm is compared with four state-of-the-art metaheuristics, i.e., cooperative and greedy monkey algorithm (CGMA) [13], modified binary particle swarm optimization (MBPSO) algorithm [30], discrete global best harmony search (DGHS) algorithm [44], and hybrid symbiotic organisms search and harmony search (HSOSHS) algorithm [21]. The comparisons are listed in Tables 14–18 where the best results are highlighted in bold. The data of [13,21,30,44] are taken from [21]. As in the HSOSHS algorithm, the maximum number of iterations is set to 25,000 and the SDHS algorithm is applied 30 times for each instance.

Table 14 shows that, on the five weakly correlated instances, the SDHS algorithm finds a new best solution on the three instances with 500, 800, and 1000 items. In terms of worst value and mean value, the SDHS algorithm obtains better results than the other four algorithms on all five instances. Table 15 shows that only the SDHS algorithm can always find the best-known solutions on all five strongly correlated instances. Table 16 shows that, on the five multiple strongly correlated instances, the SDHS algorithm finds a new best solution on the two instances with 800 and 1000 items. Except the instance with 1000 items, the SDHS algorithm can always find best-known solutions on the other four multiple strongly correlated instances. Table 17 shows that the HSOSHS algorithm and the SDHS algorithm can always find best-known solutions on all profit ceiling instances. Table 18 shows that the SDHS algorithm finds a new best solution on all the four uncorrelated instances. In terms of best value, worst value, mean value, median value, and standard deviation, the SDHS algorithm obtains better results than the other four algorithms on all four instances.

Table 14. Comparison of the SDHS algorithm with four metaheuristics on large-scale weakly correlated instances.

Dim	Opt	Method	Best	Worst	Mean	Median	Std	Time (s)
200	8714	CGMA	8714	8710	8713.00	8714	1.2318	-
		MBPSO	8699	8685	8692.13	8692	3.2772	-
		DGHS	8672	8642	8652.80	8651	6.3702	-
		HSOSHS	8714	8710	8711.47	8711	1.1366	-
		SDHS	8714	8714	8714	8714	0	0.06
300	12,632	CGMA	12,632	12,626	12,629.50	12,629	1.5702	-
		MBPSO	12,588	12,570	12,577.27	12,576	4.6382	-
		DGHS	12,532	12,502	12,516.13	12,516.5	8.4598	-
		HSOSHS	12,632	12,626	12,629.97	12,630	1.4735	-
		SDHS	12,632	12,632	12,632	12,632	0	0.08
500	22,147	CGMA	22,141	22,128	22,135.40	22,136	3.5292	-
		MBPSO	22,053	22,010	22,025.57	22,024.5	10.2071	-
		DGHS	21,965	21,917	21,934.33	21,930.5	12.1324	-
		HSOSHS	22,147	22,131	22,141.47	22,142	3.7207	-
		SDHS	22,148	22,148	22,148	22,148	0	0.14
800	35,749	CGMA	35,734	35,695	35,717.03	35,719	9.9186	-
		MBPSO	35,564	35,499	35,516.83	35,512.5	15.7854	-
		DGHS	35,469	35,368	35,394.27	35,391	20.2006	-
		HSOSHS	35,749	35,730	35,739.23	35,740	5.7095	-
		SDHS	35,762	35,762	35,762	35,762	0	0.26
1000	44,063	CGMA	44,042	43,996	44,018.03	44,015.5	11.3820	-
		MBPSO	43,788	43,713	43,741.30	43,740	15.7045	-
		DGHS	43,626	43,571	43,598.87	43,598.5	15.4334	-
		HSOSHS	44,063	44,024	44,047.87	44,047.5	7.9816	-
		SDHS	44,092	44,092	44,092	44,092	0	0.36

Note: Bold numbers indicate the algorithm found the relatively best solution.

Table 15. Comparison of the SDHS algorithm with four metaheuristics on large-scale strongly correlated instances.

Dim	Opt	Method	Best	Worst	Mean	Median	Std	Time (s)
200	9775	CGMA	9775	9775	9775	9775	0	-
		MBPSO	9775	9775	9775	9775	0	-
		DGHS	9775	9766	9772.77	9773	2.5955	-
		HSOSHS	9775	9775	9775	9775	0	-
		SDHS	9775	9775	9775	9775	0	0.07

Table 15. Cont.

Dim	Opt	Method	Best	Worst	Mean	Median	Std	Time (s)
300	14,760	CGMA	14,760	14,750	14,751.00	14,750	3.0513	-
		MBPSO	14,760	14,750	14,751.77	14,750	3.2872	-
		DGHS	14,750	14,738	14,743.00	14,740	4.1936	-
		HSOSHS	14,760	14,750	14,759.33	14,760	2.5371	-
		SDHS	14,760	14,760	14,760	14,760	0	0.1
500	25,597	CGMA	25,587	25,577	25,578.33	25,577	3.4575	-
		MBPSO	25,587	25,577	25,578.57	25,577	3.2872	-
		DGHS	25,567	25,554	25,561.40	25,563	4.4613	-
		HSOSHS	25,597	25,587	25,590.67	25,587	4.9013	-
		SDHS	25,597	25,597	25,597	25,597	0	0.19
800	39,940	CGMA	39,920	39,910	39,914.33	39,910	5.0401	-
		MBPSO	39,920	39,900	39,904.17	39,901	5.0860	-
		DGHS	39,894	39,867	39,877.20	39,877.5	5.8804	-
		HSOSHS	39,940	39,930	39,931.33	39,930	3.4575	-
		SDHS	39,940	39,940	39,940	39,940	0	0.28
1000	48,763	CGMA	48,743	48,713	48,728.27	48,732	6.2419	-
		MBPSO	48,723	48,703	48,712.40	48,712	4.4303	-
		DGHS	48,699	48,671	48,679.67	48,678	8.3101	-
		HSOSHS	48,763	48,743	48,752.33	48,753	3.6515	-
		SDHS	48,763	48,763	48,763	48,763	0	0.35

Note: Bold numbers indicate the algorithm found the relatively best solution.

Table 16. Comparison of the SDHS algorithm with four metaheuristics on large-scale multiple strongly correlated instances.

Dim	Opt	Method	Best	Worst	Mean	Median	Std	Time (s)
300	17,259	CGMA	17,259	17,239	17,253.67	17,259	8.9955	-
		MBPSO	17,259	17,257	17,258.83	17,259	0.5307	-
		DGHS	17,239	17,219	17,234.13	17,236	5.7878	-
		HSOSHS	17,259	17,259	17,259	17,259	0	-
		SDHS	17,259	17,259	17,259	17,259	0	0.09
500	29,775	CGMA	29,755	29,735	29,752.33	29,755	6.9149	-
		MBPSO	29,755	29,735	29,752.13	29,755	6.1180	-
		DGHS	29,735	29,697	29,714.93	29,712	9.7660	-
		HSOSHS	29,775	29,755	29,772.33	29,775	6.9149	-
		SDHS	29,775	29,775	29,775	29,775	0	0.15
800	47,153	CGMA	47,133	47,113	47,120.33	47,113	9.8027	-
		MBPSO	47,113	47,092	47,106.57	47,111	7.8111	-
		DGHS	47,079	47,033	47,053.30	47,052	11.0365	-
		HSOSHS	47,153	47,133	47,151.67	47,153	5.0742	-
		SDHS	47,173	47,153	47,167.83	47,172	7.1609	0.28
1000	59,575	CGMA	59,555	59,495	59,516.33	59,515	12.7937	-
		MBPSO	59,513	59,475	59,490.33	59,492	8.8213	-
		DGHS	59,455	59,403	59,427.83	59,431	12.6248	-
		HSOSHS	59,575	59,555	59,566.33	59,575	10.0801	-
		SDHS	59,595	59,575	59,578.2	59,575	5.5172	0.37
1200	69,122	CGMA	69,062	69,021	69,037.67	69,042	13.9243	-
		MBPSO	69,012	68,980	68,996.53	68,998.5	6.9319	-
		DGHS	68,949	68,892	68,916.10	68,917.5	12.8529	-
		HSOSHS	69,122	69,082	69,103.33	69,102	8.9955	-
		SDHS	69,122	69,122	69,122	69,122	0	0.43

Note: Bold numbers indicate the algorithm found the relatively best solution.

Table 17. Comparison of the SDHS algorithm with four metaheuristics on large-scale profit ceiling instances.

Dim	Opt	Method	Best	Worst	Mean	Median	Std	Time (s)
300	13,143	CGMA	13,143	13,140	13,142.20	13,143	1.3493	-
		MBPSO	13,143	13,140	13,141.40	13,140	1.5222	-
		DGHS	13,137	13,131	13,134.30	13,134	1.2077	-
		HSOSHS	13,143	13,143	13,143	13,143	0	-
		SDHS	13,143	13,143	13,143	13,143	0	0.09
500	21,069	CGMA	21,069	21,063	21,068.00	21,069	1.6400	-
		MBPSO	21,069	21,063	21,066.10	21,066	1.4704	-
		DGHS	21,060	21,054	21,056.00	21,057	1.8194	-
		HSOSHS	21,069	21,069	21,069	21,069	0	-
		SDHS	21,069	21,069	21,069	21,069	0	0.16
800	34,227	CGMA	34,227	34,221	34,223.90	34,224	2.2947	-
		MBPSO	34,221	34,215	34,216.40	34,215	2.1909	-
		DGHS	34,209	34,197	34,201.50	34,200	2.4600	-
		HSOSHS	34,227	34,227	34,227	34,227	0	-
		SDHS	34,227	34,227	34,227	34,227	0	0.27
1000	42,108	CGMA	42,108	42,108	42,108	42,108	0	-
		MBPSO	42,108	42,099	42,104.10	42,105	2.1066	-
		DGHS	42,096	42,084	42,088.00	42,087	2.9827	-
		HSOSHS	42,108	42,108	42,108	42,108	0	-
		SDHS	42,108	42,108	42,108	42,108	0	0.36
1200	51,585	CGMA	51,585	51,582	51,583.80	51,585	1.4948	-
		MBPSO	51,576	51,570	51,572.20	51,573	2.0745	-
		DGHS	51,558	51,549	51,553.80	51,552	2.4410	-
		HSOSHS	51,585	51,585	51,585	51,585	0	-
		SDHS	51,585	51,585	51,585	51,585	0	0.45

Note: Bold numbers indicate the algorithm found the relatively best solution.

Table 18. Comparison of the SDHS algorithm with six metaheuristics on large-scale uncorrelated instances.

Dim	Opt	Method	Best	Worst	Mean	Median	Std	Time (s)
3000	1914.62	CGMA	1905.08	1901.03	1902.49	1902.28	1.0721	-
		MBPSO	1887.61	1884.12	1885.70	1885.38	0.9509	-
		DGHS	1915.72	1874.84	1878.48	1876.60	8.8626	-
		HSOSHS	1914.62	1912.41	1913.55	1913.65	0.5508	-
		SDHS	1918.25	1918.24	1918.24	1918.24	0.0014	1.26
5000	3198.62	CGMA	3174.57	3167.25	3171.66	3171.79	1.5624	-
		MBPSO	3149.96	3144.21	3146.81	3146.76	1.1317	-
		DGHS	3136.95	3130.92	3133.70	3133.39	1.8258	-
		HSOSHS	3198.62	3193.24	3195.30	3195.17	1.4499	-
		SDHS	3210.52	3210.47	3210.50	3210.50	0.0096	2.52
7000	4444.40	CGMA	4410.22	4399.96	4406.19	4406.11	2.4319	-
		MBPSO	4379.24	4373.04	4375.35	4375.22	1.9022	-
		DGHS	4366.02	4356.18	4359.12	4358.52	2.5196	-
		HSOSHS	4444.40	4436.23	4438.50	4438.28	2.1447	-
		SDHS	4469.43	4469.24	4469.34	4469.34	0.0348	4.16
10000	6340.18	CGMA	6294.93	6284.56	6288.19	6287.12	2.7812	-
		MBPSO	6251.53	6246.10	6249.05	6249.12	1.6007	-
		DGHS	6235.64	6227.47	6231.33	6230.79	2.0268	-
		HSOSHS	6340.18	6327.39	6333.76	6334.03	3.2233	-
		SDHS	6387.25	6386.69	6387.02	6387.00	0.1138	7.19

Note: Bold numbers indicate the algorithm found the relatively best solution.

According to the Friedman test results in Table 19, SDHS achieves the highest ranking among the evaluated algorithms. It significantly outperforms CGMA, MBPOS, and DGHS, as confirmed by Holm's post hoc test ($p < 0.05$). Although the difference with HSOSHS is

not statistically significant, SDHS maintains a superior rank and consistently performs at least as well as all as the compared algorithms across all cases.

Table 19. The performance summary between SDHS, CGMA, MBPOS, DGHS, and HSOSHS; the average ranks are computed by the Friedman test and the significance is assessed by Holm’s post hoc method.

Algorithm	Rank	Gap	<i>p</i> -Value	<i>p</i> -Holm	Sum of + / = / −
SDHS	1.21	-	-	-	-
CGMA	3.02	0.257	6.83×10^{-4}	1.37×10^{-3}	0/2/22
MBPOS	3.81	0.491	1.16×10^{-7}	3.47×10^{-7}	0/1/23
DGHS	5	0.685	1.11×10^{-15}	4.44×10^{-15}	0/0/24
HSOSHS	1.96	0.111	4.70×10^{-1}	4.70×10^{-1}	0/6/18

Note: Bold values in the *p* columns indicate statistical significance ($p < 0.05$).

6. Conclusions

This paper presents a simplified discrete HS algorithm for solving the 0-1 KP. The time complexity of the SDHS is $O(hms \times n + m \times (n + \log_2 hms))$, where m represents the number of iterations of the algorithm, n is the number of items in the problem instance, and hms is the capacity of the harmony memory. The proposed SDHS algorithm has three unique features: (1) the random search operator is discarded and there is only one intrinsic parameter, i.e., harmony memory size, which greatly simplifies the parameter tuning; (2) in the memory consideration operator, the profit–weight ratio is used to guide the processing order of items, which can effectively improve its intensification ability; (3) the pitch adjustment operator is implemented at the solution level and is greedily guided by the profit of items, which can provide an extra complement for the profit–weight ratio-guided memory consideration operator. Experiment results confirm the positive effect of these three features on the performance of the SDHS algorithm. Furthermore, the performance of the SDHS algorithm was tested on three sets of large-scale 0-1KP instances and compared with thirteen state-of-the-art metaheuristics. The results of the comparative experiments confirm that the SDHS algorithm performs better than the 13 compared algorithms. The SDHS algorithm is not only very simple but is also very effective for the 0-1KP. A meaningful research direction is to study whether the strategies used by the SDHS algorithm are still effective and efficient for other variants of KP.

However, SDHS may still exhibit certain limitations: (1) The strict binary nature (each item is either 0 or 1) of the decision variables for the 0-1 knapsack problem may permit the SDHS to maintain adequate solution diversity through its harmony memory and pitch adjustment operator, even after discarding the random search operator. However, this simplification might pose challenges when extending the SDHS to other combinatorial optimization problems, such as those involving graph theory or variables with larger domains. In such instances, the algorithm could be susceptible to premature convergence towards local optima due to the absence of a random jumping mechanism, a crucial component for exploring disparate regions of the search space. (2) The efficacy of the algorithm is significantly dependent on the effectiveness of the two selected heuristics, the profit–weight ratio and the profit. Although these heuristics are generally quite effective for the knapsack problem, their optimality across all possible scenarios cannot be guaranteed. There may exist deceptive or specifically constructed knapsack instances in which these greedy rules could steer the search towards suboptimal regions.

Author Contributions: F.Z.: Writing—original draft, Software. K.C.: Writing—review and editing, Validation, Supervision, Methodology, Investigation, Formal analysis. K.Y.: Writing—review and editing, Investigation, Formal analysis. N.L.: Writing—review and editing, Investigation, Formal

analysis. Y.L.: Writing—review and editing, Writing—original draft, Validation, Methodology, Conceptualization. Y.Z.: Writing—review and editing, Supervision, Methodology, Funding acquisition, Conceptualization. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported by Nature Science Foundation of Fujian Province of P. R. China (No. 2023J01078) and Special Fund for Scientific and Technological Innovation of Fujian Agriculture and Forestry University of P. R. China (No. KFB23192).

Data Availability Statement: Our research datas are shared in <https://github.com/1497135676/SDHS4KP> (accessed on 8 May 2025, our java version: 1.8.0_172, IntelliJ IDEA: 2024.1).

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

References

1. Taillandier, F.; Fernandez, C.; Ndiaye, A. Real estate property maintenance optimization based on multiobjective multidimensional knapsack problem. *Comput.-Aided Civ. Infrastruct. Eng.* **2017**, *32*, 227–251. [\[CrossRef\]](#)
2. Karaboghossian, T.; Zito, M. Easy knapsacks and the complexity of energy allocation problems in the smart grid. *Optim. Lett.* **2018**, *12*, 1553–1568. [\[CrossRef\]](#)
3. Brandt, F.; Nickel, S. The air cargo load planning problem—A consolidated problem definition and literature review on related problems. *Eur. J. Oper. Res.* **2019**, *275*, 399–410. [\[CrossRef\]](#)
4. Liu, J.; Bi, J.; Xu, S. An improved attack on the basic Merkle–Hellman Knapsack cryptosystem. *IEEE Access* **2019**, *7*, 59388–59393. [\[CrossRef\]](#)
5. Yates, J.; Lakshmanan, K. A constrained binary knapsack approximation for shortest path network interdiction. *Comput. Ind. Eng.* **2011**, *61*, 981–992. [\[CrossRef\]](#)
6. Tavana, M.; Keramatpour, M.; Santos-Arteaga, F.J.; Ghorbaniane, E. A fuzzy hybrid project portfolio selection method using data envelopment analysis, TOPSIS and integer programming. *Expert Syst. Appl.* **2015**, *42*, 8432–8444. [\[CrossRef\]](#)
7. Wang, X.; He, Y. Evolutionary Algorithms for Knapsack Problems. *J. Softw.* **2017**, *28*, 1–16.
8. Gherboudj, A.; Layeb, A.; Chikhi, S. Solving 0-1 knapsack problems by a discrete binary version of cuckoo search algorithm. *Int. J. Bio-Inspired Comput.* **2012**, *4*, 229–236. [\[CrossRef\]](#)
9. Feng, Y.; Wang, G.G.; Gao, X.Z. A novel hybrid cuckoo search algorithm with global harmony search for 0-1 knapsack problems. *Int. J. Comput. Intell. Syst.* **2016**, *9*, 1174–1190. [\[CrossRef\]](#)
10. Bhattacharjee, K.K.; Sarmah, S.P. Modified swarm intelligence based techniques for the knapsack problem. *Appl. Intell.* **2017**, *46*, 158–179. [\[CrossRef\]](#)
11. Nguyen, P.H.; Wang, D.; Truong, T.K. A Novel Binary Social Spider Algorithm for 0-1 Knapsack Problem. *Int. J. Innov. Comput. Inf. Control* **2017**, *13*, 2039–2049.
12. Kulkarni, A.J.; Shabir, H. Solving 0-1 knapsack problem using cohort intelligence algorithm. *Int. J. Mach. Learn. Cybern.* **2016**, *7*, 427–441. [\[CrossRef\]](#)
13. Zhou, Y.; Chen, X.; Zhou, G. An improved monkey algorithm for a 0-1 knapsack problem. *Appl. Soft Comput.* **2016**, *38*, 817–830. [\[CrossRef\]](#)
14. Wu, H.; Zhang, F.; Zhan, R.; Wang, S.; Zhang, C. A binary wolf pack algorithm for solving 0-1 knapsack problem. *Syst. Eng. Electron.* **2014**, *36*, 1660–1667.
15. Yassien, E.; Masadeh, R.; Alzaqebah, A.; Shaheen, A. Grey Wolf Optimization Applied to the 0/1 Knapsack Problem. *Int. J. Comput. Appl.* **2017**, *169*, 11–15. [\[CrossRef\]](#)
16. Gao, Y.; Zhang, F.; Zhao, Y.; Li, C. Quantum-Inspired Wolf Pack Algorithm to Solve the 0-1 Knapsack Problem. *Math. Probl. Eng.* **2018**, *2018*, 5327056. [\[CrossRef\]](#)
17. Erdoğan, F.; Karakoyun, M.; Gülcü, Ş. An effective binary dynamic grey wolf optimization algorithm for the 0-1 knapsack problem. *Multimed. Tools Appl.* **2024**. [\[CrossRef\]](#)
18. Wang, Y.; Wang, W. Quantum-inspired differential evolution with grey wolf optimizer for 0-1 knapsack problem. *Mathematics* **2021**, *9*, 1233. [\[CrossRef\]](#)
19. Rizk-Allah, R.M.; Hassanien, A.E. New binary bat algorithm for solving 0-1 knapsack problem. *Complex Intell. Syst.* **2018**, *4*, 31–53. [\[CrossRef\]](#)
20. Yampolskiy, R.V.; El-Barkouky, A. Wisdom of artificial crowds algorithm for solving NP-hard problems. *Int. J. Bio-Inspired Comput.* **2011**, *3*, 358–369. [\[CrossRef\]](#)
21. Wu, H.; Zhou, Y.; Luo, Q. Hybrid symbiotic organisms search algorithm for solving 0-1 knapsack problem. *Int. J. Bio-Inspired Comput.* **2018**, *12*, 23–53. [\[CrossRef\]](#)

22. Abdel-Basset, M.; Zhou, Y. An elite opposition-flower pollination algorithm for a 0-1 knapsack problem. *Int. J. Bio-Inspired Comput.* **2018**, *11*, 46–53. [[CrossRef](#)]
23. Truong, T.K.; Li, K.; Xu, Y. Chemical reaction optimization with greedy strategy for the 0-1 knapsack problem. *Appl. Soft Comput.* **2013**, *13*, 1774–1780. [[CrossRef](#)]
24. Truong, T.K.; Li, K.; Xu, Y.; Ouyang, A.; Nguyen, T.T. Solving 0-1 knapsack problem by artificial chemical reaction optimization algorithm with a greedy strategy. *J. Intell. Fuzzy Syst.* **2015**, *28*, 2179–2186. [[CrossRef](#)]
25. Zhao, J.; Huang, T.; Pang, F.; Liu, Y. Genetic Algorithm Based on Greedy Strategy in the 0-1 Knapsack Problem. In Proceedings of the Third International Conference on Genetic and Evolutionary Computing, Guilin, China, 14–16 October 2009; pp. 105–107.
26. Umbarkar, A.J.; Joshi, M.S. 0/1 knapsack problem using diversity based dual population genetic algorithm. *Int. J. Intell. Syst. Appl.* **2014**, *6*, 34–40. [[CrossRef](#)]
27. Feng, Y.; Wang, G.G.; Deb, S.; Lu, M.; Zhao, X.J. Solving 0-1 knapsack problem by a novel binary monarch butterfly optimization. *Neural Comput. Appl.* **2017**, *28*, 1619–1634. [[CrossRef](#)]
28. Feng, Y.; Wang, G.G.; Dong, J.; Wang, L. Opposition-based learning monarch butterfly optimization with Gaussian perturbation for large-scale 0-1 knapsack problem. *Comput. Electr. Eng.* **2018**, *67*, 454–468. [[CrossRef](#)]
29. Feng, Y.; Yang, J.; Wu, C.; Lu, M.; Zhao, X.J. Solving 0-1 knapsack problems by chaotic monarch butterfly optimization algorithm with Gaussian mutation. *Memetic Comput.* **2018**, *10*, 135–150. [[CrossRef](#)]
30. Bansal, J.C.; Deep, K. A Modified Binary Particle Swarm Optimization for Knapsack Problems. *Appl. Math. Comput.* **2012**, *218*, 11042–11061. [[CrossRef](#)]
31. Abdel-Basset, M.; El-Shahat, D.; Sangaiah, A.K. A modified nature inspired meta-heuristic whale optimization algorithm for solving 0-1 knapsack problem. *Int. J. Mach. Learn. Cybern.* **2019**, *10*, 495–514. [[CrossRef](#)]
32. Bhattacharjee, K.K.; Sarmah, S.P. Shuffled frog leaping algorithm and its application to 0/1 knapsack problem. *Appl. Soft Comput.* **2014**, *19*, 252–263. [[CrossRef](#)]
33. Zhang, J.; Jiang, W.; Zhao, K. An improved Shuffled frog-leaping algorithm to solving 0-1 knapsack problem. *IEEE Access* **2024**, *12*, 148155–148166. [[CrossRef](#)]
34. Abdollahzadeh, B.; Barshandeh, S.; Javadi, H.; Epicoco, N. An enhanced binary slime mould algorithm for solving the 0-1 knapsack problem. *Eng. Comput.* **2022**, *38*, 3423–3444. [[CrossRef](#)]
35. Shu, Z.; Ye, Z.; Zong, X.; Liu, S.; Zhang, D.; Wang, C.; Wang, M. A modified hybrid rice optimization algorithm for solving 0-1 knapsack problem. *Appl. Intell.* **2022**, *52*, 5751–5769. [[CrossRef](#)]
36. Abdel-Basset, M.; Mohamed, R.; Saber, S.; Hezam, I.M.; Sallam, K.M.; Hameed, I.A. Binary metaheuristic algorithms for 0-1 knapsack problems: Performance analysis, hybrid variants, and real-world application. *J. King Saud Univ.-Comput. Inf. Sci.* **2024**, *36*, 102093. [[CrossRef](#)]
37. Yildizdan, G.; Baş, E. A novel binary artificial jellyfish search algorithm for solving 0-1 knapsack problems. *Neural Process. Lett.* **2023**, *55*, 8605–8671. [[CrossRef](#)]
38. Abdel-Basset, M.; Mohamed, R.; Hezam, I.M.; Sallam, K.M.; Alshamrani, A.M.; Hameed, I.A. A novel binary Kepler optimization algorithm for 0-1 knapsack problems: Methods and applications. *Alex. Eng. J.* **2023**, *82*, 358–376. [[CrossRef](#)]
39. Zou, D.; Gao, L.; Li, S.; Wu, J. Solving 0-1 knapsack problem by a novel global harmony search algorithm. *Appl. Soft Comput.* **2011**, *11*, 1556–1564. [[CrossRef](#)]
40. Layeb, A. A hybrid quantum inspired harmony search algorithm for 0-1 optimization problems. *J. Comput. Appl. Math.* **2013**, *253*, 14–25. [[CrossRef](#)]
41. Wang, L.; Yang, R.; Xu, Y.; Niu, Q.; Pardalos, P.M.; Fei, M. An improved adaptive binary Harmony Search algorithm. *Inf. Sci.* **2013**, *232*, 58–87. [[CrossRef](#)]
42. Ouyang, H.B.; Gao, L.Q.; Kong, X.Y.; Liu, H.Z. A binary modified harmony search algorithm for 0-1 knapsack problem. *Control Decis.* **2014**, *29*, 1174–1180.
43. Tuo, S.; Yong, L.; Deng, F.A. A novel harmony search algorithm based on teaching-learning strategies for 0-1 knapsack problems. *Sci. World J.* **2014**, *2014*, 637412. [[CrossRef](#)] [[PubMed](#)]
44. Xiang, W.L.; An, M.Q.; Li, Y.Z.; He, R.C.; Zhang, J.F. A novel discrete global-best harmony search algorithm for solving 0-1 knapsack problems. *Discret. Dyn. Nat. Soc.* **2014**, *2014*, 573731. [[CrossRef](#)]
45. Kong, X.; Gao, L.; Ouyang, H.; Li, S. A simplified binary harmony search algorithm for large scale 0-1 knapsack problems. *Expert Syst. Appl.* **2015**, *42*, 5337–5355. [[CrossRef](#)]
46. Liu, K.; Ouyang, H.; Li, S.; Gao, L. A Hybrid Harmony Search Algorithm with Distribution Estimation for Solving the 0-1 Knapsack Problem. *Math. Probl. Eng.* **2022**, *2022*, 8440165. [[CrossRef](#)]
47. Zhan, S.; Zhang, Z.; Wang, L.; Zhong, Y. List-Based Simulated Annealing Algorithm With Hybrid Greedy Repair and Optimization Operator for 0-1 Knapsack Problem. *IEEE Access* **2018**, *6*, 54447–54458. [[CrossRef](#)]
48. Zhan, S.; Wang, L.; Zhang, Z.; Zhong, Y. Noising methods with hybrid greedy repair operator for 0-1 knapsack problem. *Memetic Comput.* **2020**, *12*, 37–50. [[CrossRef](#)]

49. Wu, L.; Lin, K.; Lin, X.; Lin, J. List-based threshold accepting algorithm with improved neighbor operator for 0-1 knapsack problem. *Algorithms* **2024**, *17*, 478. [\[CrossRef\]](#)
50. Feng, Y.; Jia, K.; He, Y. An improved hybrid encoding cuckoo search algorithm for 0-1 knapsack problems. *Comput. Intell. Neurosci.* **2014**, *2014*, 970456. [\[CrossRef\]](#)
51. Feng, Y.; Wang, G.G.; Feng, Q.; Zhao, X.J. An effective hybrid cuckoo search algorithm with improved shuffled frog leaping algorithm for 0-1 knapsack problems. *Comput. Intell. Neurosci.* **2014**, *2014*, 857254. [\[CrossRef\]](#)
52. Geem, Z.W.; Kim, J.H.; Loganathan, G.V. A New Heuristic Optimization Algorithm: Harmony Search. *Simulation* **2001**, *76*, 60–68. [\[CrossRef\]](#)
53. Zhang, T.; Geem, Z.W. Review of harmony search with respect to algorithm structure. *Swarm Evol. Comput.* **2019**, *48*, 31–43. [\[CrossRef\]](#)
54. Ouyang, H.; Wu, W.; Zhang, C.L.; Li, S.; Zou, D.; Liu, G. Improved harmony search with general iteration models for engineering design optimization problems. *Soft Comput.* **2019**, *23*, 10225–10260. [\[CrossRef\]](#)
55. Nazariheris, M.; Mohammadiivatloo, B.; Asadi, S.; Kim, J.; Geem, Z.W. Harmony search algorithm for energy system applications: An updated review and analysis. *J. Exp. Theor. Artif. Intell.* **2019**, *31*, 723–749. [\[CrossRef\]](#)
56. Yi, J.; Gao, L.; Li, X.; Shoemaker, C.A.; Lu, C. An on-line variable-fidelity surrogate-assisted harmony search algorithm with mul-ti-level screening strategy for expensive engineering design optimization. *Knowl. Based Syst.* **2019**, *170*, 1–19. [\[CrossRef\]](#)
57. Lenin, N.; Siva Kumar, M. Harmony search algorithm for simultaneous minimization of bi-objectives in multi-row parallel machine layout problem. *Evol. Intell.* **2021**, *14*, 1495–1522. [\[CrossRef\]](#)
58. Luo, K. A sequence learning harmony search algorithm for the flexible process planning problem. *Int. J. Prod. Res.* **2022**, *60*, 3182–3200. [\[CrossRef\]](#)
59. Gong, J.; Zhang, Z.; Liu, J.; Guan, C.; Liu, S. Hybrid algorithm of harmony search for dynamic parallel row ordering problem. *J. Manuf. Syst.* **2021**, *58*, 159–175. [\[CrossRef\]](#)
60. Geem, Z.W.; Sim, K. Parameter-setting-free harmony search algorithm. *Appl. Math. Comput.* **2010**, *217*, 3881–3889. [\[CrossRef\]](#)
61. Luo, K.; Ma, J.; Zhao, Q. Enhanced self-adaptive global-best harmony search without any extra statistic and external archive. *Inf. Sci.* **2019**, *482*, 228–247. [\[CrossRef\]](#)
62. Pisinger, D. Where are the hard knapsack problems? *Comput. Oper. Res.* **2005**, *32*, 2271–2284. [\[CrossRef\]](#)

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.