mathematics

Article

Hybrid Learning Moth Search Algorithm for Solving Multidimensional Knapsack Problems

Yanhong Feng [1,2], Hongmei Wang [3,*], Zhaoquan Cai [4,5,*], Mingliang Li [1,2] and Xi Li [1]

1 School of Information Engineering, Hebei GEO University, Shijiazhuang 050031, China
2 Intelligent Sensor Network Engineering Research Center of Hebei Province, Shijiazhuang 050031, China
3 Xinjiang Institute of Engineering, Information Engineering College, Ürümqi 830023, China
4 Shanwei Institute of Technology, Shanwei 516600, China
5 School of Computer Science and Engineering, Huizhou University, Huizhou 516007, China
* Correspondence: 320180032@xjau.edu.cn (H.W.); cai@hzu.edu.cn (Z.C.)

Abstract: The moth search algorithm (MS) is a relatively new metaheuristic optimization algorithm which mimics the phototaxis and Lévy flights of moths. Being an NP-hard problem, the 0–1 multidimensional knapsack problem (MKP) is a classical multi-constraint complicated combinatorial optimization problem with numerous applications. In this paper, we present a hybrid learning MS (HLMS) by incorporating two learning mechanisms, global-best harmony search (GHS) learning and Baldwinian learning for solving MKP. (1) GHS learning guides moth individuals to search for more valuable space and the potential dimensional learning uses the difference between two random dimensions to generate a large jump. (2) Baldwinian learning guides moth individuals to change the search space by making full use of the beneficial information of other individuals. Hence, GHS learning mainly provides global exploration and Baldwinian learning works for local exploitation. We demonstrate the competitiveness and effectiveness of the proposed HLMS by conducting extensive experiments on 87 benchmark instances. The experimental results show that the proposed HLMS has better or at least competitive performance against the original MS and some other state-of-the-art metaheuristic algorithms. In addition, the parameter sensitivity of Baldwinian learning is analyzed and two important components of HLMS are investigated to understand their impacts on the performance of the proposed algorithm.

Keywords: combinatorial optimization; multidimensional knapsack problem; metaheuristic; moth search algorithm; Baldwinian learning; global-best harmony search

MSC: 90C27

Citation: Feng, Y.; Wang, H.; Cai, Z.; Li, M.; Li, X. Hybrid Learning Moth Search Algorithm for Solving Multidimensional Knapsack Problems. *Mathematics* **2023**, *11*, 1811. https://doi.org/10.3390/math11081811

Academic Editor: Anatoliy Swishchuk

Received: 8 March 2023
Revised: 31 March 2023
Accepted: 4 April 2023
Published: 11 April 2023

1. Introduction

The multidimensional knapsack problem (MKP) [1,2] is a generalization of the 0–1 knapsack problem (0–1 KP) [3]. As a classical combinatorial optimization problem, numerous real-world applications can be modeled as MKP, such as capital budgeting [4,5], cutting stock [6], and loading problem [7,8].

Then the MKP is: given a set of n items and a set of m knapsacks ($m < n$), with c_j = profit of item j, b_i = capacity constraint of knapsack i,a_{ij} = resource consumption of item j in the ith knapsack. The goal of MKP is to select a subset of items so that the total profit of the selected items is a maximum while the total weights in each dimension i ($i = 1, 2, \ldots, m$) do not exceed the corresponding capacity b_i. Formally,

$$max \quad z = \sum_{j=1}^{n} c_j x_j \tag{1}$$

$$s.t. \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad \forall i \in \{1, 2, \ldots, m\} \tag{2}$$

$$x_j \in \{0, 1\}, \quad \forall j \in \{1, 2, \ldots, n\} \tag{3}$$

where x_j ($j = 1, 2, \ldots, n$) is a 0–1 decision variable, such that $x_j = 1$ if item j is assigned to a knapsack, $x_j = 0$ otherwise. When $m = 1$, MKP reduces to the 0–1 KP.

Known as the NP-hard combinatorial optimization problem of MKP, the conventional exact algorithms usually are unable to obtain a satisfactory solution in a reasonable time, especially for large-scale instances. It is because that combinatorial optimization problems often feature a combination of the explosion and then the search space grows exponentially with the expansion of the scale. In this case, metaheuristic algorithms are effective methods for MKP, which can obtain near-optimal solutions in a reasonable acceptable time. Representative metaheuristic algorithms for solving MKP include the two-phase tabu evolutionary algorithm (TPTEA) [9], binary particle swarm optimization (PSO) [10], quantum particle swarm optimization (QPSO) [11], diversity-preserving quantum particle swarm optimization (DQPSO*) [12], hybrid estimation of distribution algorithm (EDA) [13], memetic algorithm (MA) [14], binary grey wolf optimizer (GWO) [15], hybrid harmony search algorithm (HS) [16], binary multi-verse optimizer (MMVO) [17], cuckoo search (CS) [18], pigeon-inspired optimization algorithm (PIO) [19], binary moth search algorithm (MS) [20], sine cosine algorithm (SCA) [21], and binary slime mould algorithm (BSMA) [22]. For more information on evolutionary algorithms (EAs) [23] and exact methods for solving MKP problem, please refer to [24].

Recently, some novel metaheuristic algorithms have been proposed and used to solve various optimization problems, including continuous optimization problems and discrete optimization problems, such as differential evolution algorithm (DE) [25,26], cuckoo search algorithm (CS) [27], bat algorithm (BA) [28], krill herd (KH) [29,30], elephant herding optimization (EHO) [31], monarch butterfly optimization algorithm (MBO) [32,33], brain storm optimization algorithm (BSO) [34], fruit fly optimizer (FFO) [35], whale optimization algorithm (WOA) [36], wind driven optimization (WDO) [37], salp swarm algorithm (SSA) [38], Harris hawks optimization (HHO) [39], artificial Jellyfish Search (JS) optimizer [40], artificial rabbits optimization (ARO) [41], mountain gazelle optimizer (MGO) [42], moth-flame optimizer (MFO) [43], and many more.

The moth search (MS) [44,45] algorithm was recently developed by Wang and inspired by the phototaxis and Lévy flights of moths. Owing to its simplicity and high search efficiency, MS has been successfully applied to solve various optimization problems, such as the cloud task scheduling problem [46], drone placement problem [47], constrained optimization problems [48], discounted {0–1} knapsack problem [49], and set-union knapsack problem [50]. Previous studies have shown that MS is effective for solving various combinatorial optimization problems, especially for various variants of KP problems [49,51]. To the best of our knowledge, there are few literatures on applying MS to solve MKP. Hence, we concentrated on the MS algorithm for solving MKP.

In recent years, the learning-based metaheuristic algorithms have been extensively reported in literature because of their effectiveness and efficiency in solving various optimization problems. The core idea is that the metaheuristic algorithm combines specific learning operators or learning mechanisms to enhance itself some learning ability, and then owning better optimization behavior. There are many learning methods, for example, deep learning [52,53], reinforcement learning [54–56], transfer learning [57], Q-learning [58], information feedback [59], orthogonal learning [34], comprehensive learning [60], Baldwinian learning, and so on.

Inspired by Darwinian evolution, evolutionary computation mainly includes the random variation of individuals and fitness selection mechanism [61]. It seems time-consuming to search randomly for good genotypes without using phenotypes. One of the possible ways to surmount this deficiency is to integrate the learning mechanism into the evolutionary search. The learning mechanism can provide a more effective search path [62]. Hence, learning-based mechanisms have been extensive researched and applied in enhancing the search performances of evolution algorithms [14,31]. The Baldwin effect, sometimes called Baldwinian learning, was widely incorporated into a variety of evolutionary computing models to enhance their performance [61,63,64]. The above works motivate us to propose a Baldwinian learning MS to solve MKP.

Harmony search (HS) [65] is metaheuristic and imitates the musical improvization process to search for a perfect state of harmony. Since proposed, HS has always attracted much attention from researchers and has been successfully applied to deal with various optimization problems. An effective variant of HS, called global-best HS (GHS), was proposed by Omran [66]. Compared with HS and other variants, GHS alleviates the problem of tuning the parameter *bw* and can work efficiently on both continuous and discrete problems. The GHS algorithm, due to its easy implementation and quick convergence, has been applied to many fields. Xiang et al. [67] proposed a discrete GHS (DGHS) for solving 0–1 KP. Keshtegar et al. [68] proposed a Gaussian GHS (GGHS) algorithm for solving numerical optimization problems. EI-Abd et al. [69] proposed an improved GHS (IGHS) algorithm to solve continuous optimization. In essence, musical performances seek to find pleasing harmony, just like the process of finding the global optimal solution through continuous learning. Based on this, GHS, as another learning scheme, is integrated into the MS for a global search.

Aiming at resolving the above issues, we propose a hybrid learning MS (HLMS), by introducing the Baldwinian learning and GHS learning mechanism. In HLMS, a novel Baldwin learning strategy based on Cauchy distribution is proposed instead of Gaussian distribution in [70]. During the evolutionary process, each individual in the whole population performs GHS learning and Baldwinian learning successively with a certain probability to reduce the time-consumption. The beneficial combination and complementarity of these two mechanisms lead HLMS to evolve towards the global optimum. Intuitively, HLMS has better search performance than the original MS because of the good balance between the exploration capacity of GHS learning and the exploitation ability of the Baldwinian learning.

The novelty and main contributions of this work include.

- A novel Baldwinian learning strategy based on Cauchy distribution is proposed, the analysis and experiment of which show this strategy is more effective than the Baldwin learning strategy based on Gaussian distribution.
- Combined with Baldwin learning, GHS is an effective global search operator to enhance the exploration ability of HLMS. Meanwhile, the pitch adjustment process based on dimensional learning can generate a large jump in the search process.
- To reduce computation costs, the proposed HLMS triggers Baldwin learning and GHS learning with a certain probability in each iteration.
- Exploration and exploitation are two common and fundamental features of any optimization method. In the evolution of HLMS, Lévy flight and GHS learning are mainly responsible for exploration, whilst flight straightly and Baldwinian mainly implement exploitation.

The rest of this paper is organized as follows. Section 2 reviews the MS algorithm, the Baldwinian learning scheme, and the GHS algorithm. In Section 3, the proposed HLMS for the MKP is introduced in detail. Extensive experiments and comparisons are conducted in Section 4. Finally, conclusions and suggestions are provided in Section 5.

2. Preliminaries

In this section, we summarize the core idea of MS, Baldwinian learning, and Global-best HS algorithm, which form the basis of the proposed HLMS framework.

2.1. Moth Search Algorithm

The moth search algorithm (MS) [44] is a new metaheuristic algorithm developed by Wang and inspired by the phototaxis and Lévy flights of moths. Based on these two behaviors, the flight straightly operator and Lévy flight operator of MS are derived, which can achieve good balance between the exploration capability and exploitation ability. Meanwhile, the whole population is subdivided into two subpopulations (named subpopulation1 and subpopulation2) based on the fitness of moth individuals. To this end, the position of offspring in subpopulation1 and subpopulation2 is updated by Lévy flight and flight straightly, respectively.

In the Lévy flight stage, the core mathematical formulation is:

$$X_i^{t+1} = X_i^t + \alpha L(s) \tag{4}$$

$$\alpha = S_{max}/t^2 \tag{5}$$

$$L(s) = \frac{(\beta-1)\Gamma(\beta-1)\sin\left(\frac{\pi(\beta-1)}{2}\right)}{\pi s^{\beta}} \tag{6}$$

where X_i^t and X_i^{t+1} denote the position vector of the ith moth at generation t and t + 1, respectively. α is the scale factor and S_{max} is the max walk step. L(s) represents the step drawn from Lévy distribution with β = 1.5 and $\Gamma(x)$ is the gamma function.

In the flight straightly stage, the ith individual in subpopulation2 is considered to fly in a straight line towards the light source. The mathematical model of the flight straightly operator is formulated as follows:

$$X_i^{t+1} = \begin{cases} \lambda \times (X_i^t + \varphi \times (X_{best}^t - X_i^t)) & if\ \ rand > 0.5 \\ \lambda \times (X_i^t + \frac{1}{\varphi} \times (X_{best}^t - X_i^t)) & else \end{cases} \tag{7}$$

where λ is the scale factor, which is used to control the convergence speed of the algorithm and improve population diversity. λ is set to a random number drawn by the standard uniform distribution. The acceleration factor φ is set to golden ratio (0.618). X_{best}^t is the best moth individual at generation t. rand returns a random number that is uniformly distributed in (0, 1).

The pseudo code of MS is shown in Algorithm 1.

Algorithm 1. Moth search algorithm

Begin
Step 1: Initialization.
Set the maximum iteration number MaxGen and iteration counter G = 1; Initialize the parameters max walk step S_{max}, the index β, and acceleration factor φ.
According to uniform distribution, the population with *NP* individuals is randomly initialized.
Step 2: Fitness calculation.
Compute the initial objective function values of each individual according to their position.
Memory the best individual (denotes as X_{best}).
Step 3: While (G < *MaxGen*) **do**
Divide the whole population into two subpopulations with equal size: subpopulation1 and subpopulation2, based on their fitness.
Update subpopulation1 by using Lévy flight operator (Equation (4)).
Update subpopulation2 by using flight straightly operator (Equation (7)).
Evaluate the objective function values of each individual and update X_{best}.
$G = G + 1$.
Sort the population by fitness.
Step 4: End while
Step 5: Output: the best results.
End.

2.2. Baldwin Effect and Baldwinian Learning

The interactive way of learning and evolution was first proposed by Baldwin, known as the Baldwin effect. Hence, different Baldwinian learning models have been proposed based on the Baldwin effect. Hinton and Nowlan [62] found that it was difficult to find the optimal solution of more complex problems only by an evolutionary algorithm. However, when combined with Baldwinian learning, the performance of the hybrid algorithm can be effectively improved.

Generally speaking, Baldwinian learning is a type of local search strategy in an evolutionary algorithm. The Baldwinian learning mechanism was first combined with the clonal selection algorithm (CSA) by Gong et al. [61] to improve the performance of BCSA. Based on this, Peng et al. [70] proposed four Baldwinian learning strategies inspired by the trial vector generating strategy of differential evolution (DE).

In evolutionary computation, Cauchy mutation and Gaussian mutation are two popular and effective mutation techniques [71]. The characteristic of Gaussian mutation is to speed up the local convergence, and Cauchy mutation is better at escaping from local optimum. However, compared with Gaussian mutation, Cauchy mutation is insensitive to mutation step size and can achieve the acceptable performance.

According to the above analysis, the newly designed Baldwinian learning operator based on Cauchy distribution is proposed in HLMS. The mathematical expression is as follows:

$$Y_i = X_{r1} + c \cdot (X_{r2} - X_{r3}) \tag{8}$$

where Y_i is the donor vector for each moth individual X_i from the current population after applying Baldwinian learning. X_{r1}, X_{r2}, and X_{r3} are sampled randomly from the current population and $r1$, $r2$, and $r3$ are mutually exclusive integers randomly chosen from the range [1, NP], which are also different from the selected individual index i. The parameter c is the strength of Baldwinian learning and is a random number based on Cauchy distribution.

2.3. Global-Best HS Algorithm

GHS is a novel variant of HS and inspired by the concept of swarm intelligence of PSO [66]. The difference from the original HS is that the new harmony can mimic the best harmony in the harmony memory HM. Meanwhile, the parameter bw in HS is replaced and a social dimension is added to the GHS. In addition, the GHS dynamically updates the pitch adjusting rate PAR according to the following equation [72]:

$$PAR(t) = PAR_{min} + \frac{PAR_{max} - PAR_{min}}{NI} \times t \tag{9}$$

where PAR(t) is the pitch adjusting rate for generation t, PAR_{min} is the minimum adjusting rate, and PAR_{max} is the maximum adjusting rate. NI is the number of improvisations and t is the generation number.

The procedure of GHS is given in Algorithm 2.

Algorithm 2. The Global-best HS algorithm (GHS)

Begin
For each $i \in [1, n]$ **do** /*n is the dimension of the problem*/
If $U(0,1) \leq$ HMCR **then** /* memory consideration*/
$x_i^h = x_i^j$, where $j \sim U(1, \ldots, HMS)$
If $U(0,1) \leq$ ***PAR*** (t) **then** /*pitch adjustment*/
$x_i^h = x_k^{best}$, where *best* is the best harmony in the HM and $k \sim U(1, n)$
Else /*random selection*/
$x_i^h = LB_i + rand^*(UB_i - LB_i)$
End.

Furthermore, it should be emphasized that, the concept of dimensional learning [73] is embodied in Algorithm 2, as shown below:

$$X_i^h = X_k^{best} \tag{10}$$

In Equation (7) of MS, the same dimension j is selected in $X_{best}^t - X_i^t$ for conducting the new solution. Under this dimension, if the component value of the ith individual is similar to the best individual, the difference $X_{best}^t - X_i^t$ will be very small, especially in the later stage of evolution. This means that such a step size is not conductive to X_i jumping to a far position. If the best moth individual is local optimum, the solution hardly escapes from the local extremum. In Equation (10), the dimension index i of X^h is not equal to the dimension index k of X^{best}. Generally, the difference between two different dimensions is large. Different dimensions can carry different information.

Based on the above analysis, dimension learning is embedding into Algorithm 2 and it should be an effective global search operator.

3. The Proposed HLMS for the MKP

The proposed HLMS algorithm for MKP is inspired from the studies [66,70] and distinguishes itself with two new features. First, GHS as a powerful global search operator is introduced to enhance the exploration ability of the algorithm. Second, a new Baldwinian learning strategy by replacing Gaussian distribution with Cauchy distribution is introduced on HLMS. The proposed algorithm framework and the main components of the MKP problem are described in the following subsection.

3.1. Population Initialization

In this stage, NP moth individuals are randomly generated in the search space. The swarm X = {X(1), X(2), ... , X(NP)} is maintained and evolves, where each moth individual X(i) is a n-dimensional real-valued vector $X(i) = (x_1^i, x_2^i, \ldots, x_n^i)$ with $x_j^i \in \{-a, a\} \wedge j \in \{1, 2, \ldots, n\}$ and n is the number of objects or items. Here, a takes the value 3 or 5 in this paper. Then, each moth individual X(i) is transformed into an n-dimensional binary vector by a mapping method, which is called discrete moth $Y(i) = (y_1^i, y_2^i, \ldots, y_n^i)$ with $y_j^i \in \{0, 1\} \wedge j \in \{1, 2, \ldots, n\}$.

3.2. Solution Representation

In HLMS, an n-bit binary string consisting of 0 and 1 is used to represent a candidate solution. If the item is selected, the bit is 1, otherwise it is 0. It should be noted that the MKP is a constrained optimization problem, so the solution generated in the evolution process may be infeasible.

In this paper, a simple and effective transfer function [50] is adopted and the function expression is as follows:

$$T(x) = x \tag{11}$$

The transfer method from a real-valued variable x_i to a binary variable y_i is calculated by:

$$y_i = \begin{cases} 1, \ if \ \ T(x_i) > 0 \\ 0, \text{else} \end{cases} \tag{12}$$

3.3. Quick Repair Operator

Learning from previous research work [10,74], the HLMS algorithm also adopts a popular quick repair operator based on pseudo-utility which was proposed by Luo et al. [15]. In order to effectively apply the repair operator, the given MKP instance needs to be preprocessed. Specifically, all the items are renumbered in an ascending order based on their scaled pseudo-utility ratios σ_j [75] defined as follows:

$$\sigma_j = \frac{c_j}{\sum\limits_{i=1}^{m} \frac{a_{ij}}{b_i}}, \forall j \in \{1, 2, \ldots, n\} \tag{13}$$

More exactly, the index values of all sorted items are stored in array J [1 ... n], such that $\sigma_{J[1]} \geq \sigma_{J[2]} \geq \ldots \geq \sigma_{J[n]}$. The vectors $(c_1, c_2, \ldots, c_n)$ and $a_{i,j}$ (i = 1, 2, ... , m, j = 1, 2, ... , n) should be adjusted as well. The main framework of the quick repair procedure can be summarized in Algorithm 3.

Algorithm 3. The quick repair operator base on the scaled pseudo-utility

Begin
Step 1: Input. $X = \{x_1, x_2, \ldots, x_n\} \in \{0, 1\}^n$.
Step 2: Calculation. Calculate the current total consumption of each resource.
$r_i = \sum\limits_{j=1}^{n} a_{ij} * x_j, \forall i = 1, 2, \ldots, m.$
Step 3: Repair process.
For $j = n$ **to** 1 **do**
If $r_i \leq b_i, \forall i = 1, 2, \ldots, m$ **then**
Break.
else if $x_{J[j]} = 1$ **then**
Set $x_{J[j]} = 0$, and $r_i \leftarrow r_i - a_{iJ[j]}, \forall i = 1, 2, \ldots, m.$
End
End
Step 4: Optimization process.
For $j = 1$ **to** n **do**
If $x_{J[j]} = 0$ and $r_i + a_{iJ[j]} \leq b_i, \forall i = 1, 2, \ldots, m$ **then**
Set $x_{J[j]} = 1$ and $r_i \leftarrow r_i + a_{iJ[j]}, \forall i = 1, 2, \ldots, m.$
End
End
Step 5: Output: $X = \{x_1, x_2, \ldots, x_n\} \in \{0, 1\}^n$.
End.

Obviously, the quick repair operator of Algorithm 3 mainly consists of two phases. In the first phase, called the repair process, according to the ascending order of the scaled pseudo-utility ratios, the items are removed from the knapsack one by one until the solution is feasible. In the second phase, called the optimization process, for all the feasible solutions, greedily packed the items to be loaded into the knapsack based on the descending order of the scaled pseudo-utility ratios one by one. In this process, the feasibility of the solution needs to be maintained all the time. In brief, the first phase makes all the infeasible solutions become feasible, and the second phase enables the quality of feasible solutions better.

3.4. Procedure of HLMS for MKP

Based on the analysis above, the proposed HLMS algorithm for MKP is outlined in Algorithm 4. The algorithm framework includes the following main steps. (1) After initialization, the repaired population is divided into two subpopulations based on the fitness. (2) Subpopulation1 and subpopulation2 apply the Lévy flight operator and flight straightly operator, respectively. (3) GHS learning and Baldwinian learning are implemented in sequence to the whole population with a probability of 0.5. (4) The mapping from the real-valued vector to the binary vector is realized with a transfer function and then the repair of infeasible solutions and the optimization of feasible solutions are performed. (5) Evaluating the solution is based on the objective function and then the whole population is divided into two subpopulations. Steps (2)–(5) are repeated until the termination condition is reached.

Algorithm 4. Procedure of HLMS for MKP

Begin
Step 1: Initialization.
Set the maximum iteration number *MaxGen* and iteration counter *G* = 1; Initialize the parameters max walk step S_{max}, the index $\beta\varphi$, strength of Baldwinian learning *c*.
According to uniform distribution, the population with *NP* individuals is randomly initialized.
The transform function is used to discretize the real number vector to obtain the initial solution $X = \{x_1, x_2, \ldots, x_n\} \in \{0,1\}^n$.
Repair the initial solution by Algorithm 3.
Step 2: Fitness evaluation.
Evaluate the initial solution using the objective function of MKP.
Step 3: While g < *MaxGen* **do**
3.1 Divide the whole population into two subpopulations with equal size: subpopulation1 and subpopulation2, based on their fitness.
3.2 Update subpopulation1 by Lévy flight operator.
3.3 Update subpopulation2 by flight straightly operator.
3.4 GHS search
If U(0, 1) $\leq$ 0.5
Apply GHS algorithm on each individual X and generate the trial individual Y.
Choose the best one of X and Y to enter the next generation.
3.5 Baldwinian Learning
If U(0, 1) $\leq$ 0.5
Apply Baldwinin learning strategy on each individual X and generate the trial individual Y.
Choose the best one of X and Y to enter the next generation.
3.6 Apply transform function to obtain the potential solution of MKP.
3.7 Repair the potential solution by Algorithm 3.
3.8 Evaluate the fitness of the population and record the global best fitness.
G = *G* + 1.
3.9 Sort the population by fitness.
Step 4: End while
Step 5: Output: the best results.
End.

The algorithm framework is shown in Figure 1.

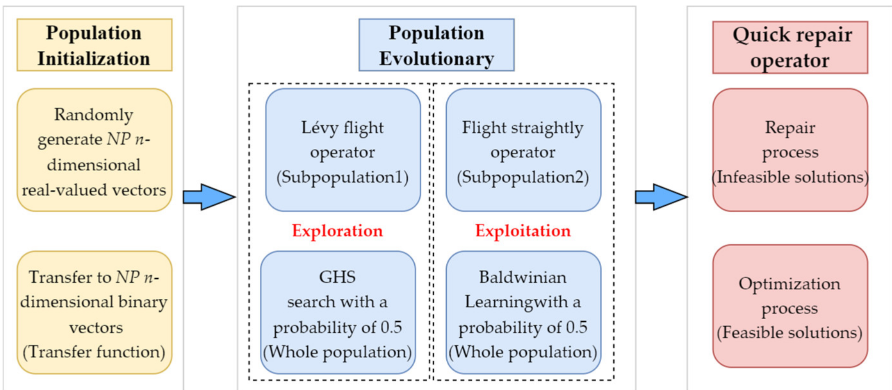

Figure 1. The framework of HLMS for MKP.

3.5. Computational Complexity of One Iteration of HLMS

The computational complexity of one iteration of HLMS based on Algorithm 4 is described as follows.

(1) The initialization of HLMS requires O(NP $\times$ n) time, where NP denotes the population size, and n is the dimension of MKP (the number of the items).
(2) The discretization process of NP moth individual costs O(NP $\times$ n) time.

(3) The quick repair operator takes O(n × m) time, where m is the constraints of the MKP instance.
(4) Fitness evaluation has O(NP) time.
(5) Lévy flight operator has O(NP_1 × n) time, where NP_1 is the number of individuals of subpopulation1.
(6) Flight straightly operator has O(NP_2 × n) time, where NP_2 is the number of individuals of subpopulation2.
(7) GHS learning requires O(NP × n) time.
(8) Baldwinian learning requires O(NP × n) time.
(9) Sort the population based on Quick sort algorithm and it takes time O(NPlogNP).

In summary, the total computational complexity is O(NP × n) per generation for fixed m.

4. Experimental Studies

To comprehensively evaluate the performance of the proposed HLMS, large numbers of experiments are implemented on the benchmark instances commonly used in the literature and a comparative study is conducted between HLMS and several populations based on optimization algorithms.

4.1. Benchmark Test Functions

In this paper, four sets of well-known benchmark data for MKP are used to test the performance of HLMS. These instances are described in [74] and available at OR-Library (http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapinfo.html, accessed on 1 July 2021).

Test set I contains 18 small-scale instances with m = 2 to 30 and n = 20 to 105, which are denoted as Sento [76], HP [77], PB [77], and Weing [4].

Test set II contains 30 medium-scale instances with m = 5 and n = 30 to 90, which are marked as Weish [7].

Test set III contains 30 large-scale instances. These instances are divided into two subsets. Subset I includes 15 instances with m = 15 and n ∈ {100, 250, 500}, which are labeled as cb1, cb2, and cb3, respectively. Subset II also includes 15 instances with m = 15 and n ∈ {100, 250, 500}, which are called as cb4, cb5, and cb6, respectively.

Test set IV contains 9 instances with m {15, 25, 50} and n ∈ {100, 200, 500, 1000, 1500}, which were created by Glover and Kochenberger and then are marked as GK.

4.2. Experimental Environment and Parameters Setting

The proposed HLMS algorithm includes several important parameters, whose values are empirically set based on the preliminary experiments and the details are recorded in Table 1.

Table 1. Settings of parameters of HLMS.

Parameters	Section	Description	Values
Smax	2.1	The max step used in Equation (5)	1.0
φ	2.1	The acceleration factor used in Equation (7)	0.618
λ	2.1	The scale factor used in Equation (7)	a random number of uniform distribution in [0, 1]
β	2.1	Parameter used in Equation (6)	1.5
c	2.2	The strength of Baldwinian learning	A random number based on Cauchy distribution
PARmax	2.3	Parameter used in Equation (9)	0.99
PARmin	2.3	Parameter used in Equation (9)	0.01
HMCR	2.3	Parameter used in Algorithm 2	0.9
NP	3.4	The population size	50
NFE	3.4	The maximal number of function evaluation	100,000

The HLMS algorithm was implemented in C language and compiled using the GNU GCC compiler. All the experiments were carried out on a computer with Intel (R) Core (TM) i7-7500 CPU (2.90 GHz and 8.00 GB RAM), running the Windows 10 operating system. HLMS was independently run 30 times for each instance to eliminate the unfairness brought by the stochastic characteristic.

To comprehensively evaluate the performance of the HLMS algorithm, fourteen MKP algorithms in the literature are selected as our comparative algorithms, which are listed as follows.

- Modified binary particle swarm optimization (MBPSO) [78];
- Chaotic binary particle swarm optimization with time-varying acceleration coefficients (CBPSOTVAC) [79];
- Binary PSO with time-varying acceleration coefficients (BPSOTVAC) [79];
- Modified multi-verse optimization (MMVO) algorithm [17];
- New binary particle swarm optimization with immunity-clonal algorithm (NPSO-CLA) [80];
- Binary gravitational search algorithm (BGSA) [81];
- Binary hybrid topology particle swarm optimization (BHTPSO) [81];
- Binary hybrid topology particle swarm optimization quadratic interpolation (BHTPSO-QI) [81];
- New binary particle swarm optimization (NBPSO) [81];
- Binary version of PSO (BPSO) [82];
- Binary version of the Harris hawks algorithm (BHHA) [22,83];
- Binary version of the salp swarm algorithm (BSSA) [84];
- Binary version of the modified whale optimization algorithm (BIWOA) [85];
- Binary version of the sin-cosine algorithm (BSCA) [86].

It should be noted that the results of the comparative algorithms are compiled from the related papers. If the result of an algorithm for a MKP instance is not available, the result of the instance is ignored. In addition, considering that different comparative algorithms are written in different programming languages, or run on different computing platforms based on different termination conditions and algorithm parameters, we focus on comparing solution quality.

For this purpose, in this paper, eight typical statistical evaluation criteria are selected to evaluate the performance of all the comparative algorithms.

- Best value (Best):

$$Best = \max(f_i),\ \text{for}\ \forall i \in [1, t] \tag{14}$$

where f_i is the fitness value for ith time. t is the total number of independent experiments.

- Worst value (Worst):

$$Worst = \min(f_i),\ \text{for}\ \forall i \in [1, t] \tag{15}$$

- Mean value (Mean):

$$Mean = \frac{1}{t}\sum\nolimits_{i=1}^{t} f_i \tag{16}$$

The mean value characterizes the centralized trend of the values of random variables. The larger the mean value, the more concentrated the results of multiple runs of the algorithm will be.

- Standard deviation (Std):

$$Std = \sqrt{\frac{1}{t}\sum\nolimits_{i=1}^{t} (f_i - mean)} \tag{17}$$

Standard deviation describes the degree of dispersion of random variable values relative to the mean value. Meanwhile, standard deviation reflects the fluctuation in the value of a random variable. In other words, stability is an important evaluation criterion of a stochastic algorithm. If the Std value is high, the stability of the algorithm is poor, otherwise, its performance is good.

- Success rate (*SR*):

$$SR = \frac{st}{t} \tag{18}$$

where st denotes the success times, that is, the number of the known theoretical optimal solution is obtained. The high success rate indicates that the algorithm has good stability and optimization performance.

- Percent deviation (*PDev*):

$$PDev = \frac{Opt - Mean}{Opt} * 100 \tag{19}$$

where Opt represents the optimal or the best-known solution. PDev reflects the degree to which the mean value deviates from the known theoretical optimal solution when the algorithm solves a single instance.

- Average error (*AE*):

$$AE = \frac{1}{N} * \sum_{i=1}^{N} \frac{(Opt - profit)}{Opt} * 100 \tag{20}$$

Average error is an indicator that reflects the general level of error between random variables and *Opt*. Here, profit can be *Best*, *Worst*, or *Mean*. *N* is the number of benchmark instances. Clearly, the smaller *AE* value indicates that the algorithm has better performance. *AE* indicates the overall performance of the algorithm for solving a set of MKP instances.

- Percentage gap (*Gap*):

$$Gap = \frac{Opt - Best}{Opt} * 100 \tag{21}$$

Similar to *AE*, for the maximization problem, the smaller the *Gap* is, the better the performance of the algorithm is. *Gap* investigates the performance of the algorithm to solve a single MKP instance.

Moreover, to determine whether there are significance differences between HLMS and other algorithms, the *p*-value based on the nonparametric Wilcoxon signed ranks test at the 95% confidence level is reported as well. Note that a *p*-value less than 0.05 represents that there exists a significant difference between the paired compared results. All the statistical results have been performed by the statistical software R language.

4.3. Comparisons on the Small-Scale Test Set

The proposed HLMS is first substantiated based on the 18 small-scale instances of Test set I and the experimental result is listed in Table 2, along with the available results of the comparative algorithms. In Table 2, the first three columns record the names of instances, the dimension information (n is the number of items, and m is the number of knapsacks), and the known optimum results (Opt), respectively. In addition, aggregate data are recorded at the bottom of the table. #Opt shows the number of the best-known solution obtained by the corresponding algorithm. #SR and #Std represent the number of instances for which the corresponding algorithm obtained a better result in terms of SR and Std among the comparative algorithms. MSR is the mean value of success rate and the ranks in descending order on the MSR are provided. Besides, the best findings among the comparison results are indicated in bold.

Table 2. The results of HLMS with 3 comparative algorithms for Test set I.

Prob.	$n \times m$	Opt.	MS		HLMS		MBPSO		CBPSOTVAC	
			SR	*Std*	*SR*	*Std*	*SR*	*Std*	*SR*	*Std*
Sento1	60 × 30	7772	0.17	54.46	**0.90**	**33.73**	0.16	43.23	0.39	357.78
Sento2	60 × 30	8722	0.00	27.79	**0.47**	**4.72**	0.03	18.80	0.20	101.03
HP1	28 × 4	3418	0.13	23.11	**0.40**	19.96	0.10	25.52	0.38	**10.69**
HP2	35 × 4	3186	0.00	26.47	0.40	32.70	0.11	39.15	**0.59**	**21.35**
PB1	27 × 4	3090	0.03	30.89	**0.43**	17.16	0.11	24.32	0.40	**10.52**
PB2	34 × 4	3186	0.00	19.95	**0.70**	**13.83**	0.16	39.31	0.51	18.73
PB4	29 × 2	95,168	0.00	894.68	0.30	1521.73	0.27	1803	**0.84**	**875.1**
PB5	20 × 10	2139	0.17	23.88	0.70	21.26	0.08	24.36	**0.80**	**6.83**
PB6	40 × 30	776	0.43	24.27	**0.80**	**17.28**	0.28	29.12	0.54	40.17
PB7	37 × 30	1035	0.03	3.86	**0.50**	**5.83**	0.05	16.29	0.40	24.25
Weing1	28 × 2	14,1278	0.90	214.94	**0.93**	**89.77**	0.82	250.43	0.92	281.98
Weing2	28 × 2	130,883	0.30	5731.34	**0.97**	**29.21**	0.65	314.08	0.88	545.50
Weing3	28 × 2	95,677	0.13	3767.42	**0.80**	1996.82	0.11	876.78	0.75	**672.42**
Weing4	28 × 2	119,337	0.73	1329.64	0.37	711.43	0.76	1270.80	**0.97**	**378.58**
Weing5	28 × 2	98,796	0.23	2671.33	**1.00**	**0.00**	0.52	1923.5	0.94	572.82
Weing6	28 × 2	130,623	0.23	164.95	**0.97**	**71.20**	0.36	322.40	**0.97**	343.45
Weing7	105 × 2	1,095,445	0.00	**482.74**	0.00	2872.29	**0.02**	1130.60	0.00	30,020.00
Weing8	105 × 2	624,319	**0.33**	1966.37	0.03	**1135.87**	0.03	4704.30	0.20	75,169.00
	#Opt		13		17		**18**		17	
	#SR		0		**12**		1		5	
	#Std		1		**10**		0		7	
	MSR		0.21		0.54		0.26		**0.59**	
	Rank of *MSR*		4		2		3		**1**	
	p-value		0.001	0.029			0.001	0.663	0.641	0.896

From Table 2, the proposed HLMS is able to obtain the known optimum solution for almost all the 18 instances except for Weing7. However, MS can only reach the known optimum solution for 13 instances. Considering the #SR, HLMS performs much better than the competing algorithms. In terms of MSR, HLMS is slightly worse than CBPSOTVAC and ranks two. Moreover, the clear superiority of HLMS is established in comparison with MS in terms of all evaluation criteria. Therefore, we can conclude that it is beneficial to use the GHS global search algorithm combined with the Baldwinian learning strategy. In terms of the *p*-value of SR, the difference between HLMS and MS, HLMS and MBPSO is statistically significant (p-value < 0.05). However, there are no significant differences in Std among the latter two groups (p-value > 0.05).

The related box plots are given in Figure 2 in terms of SR. As can be seen from Figure 2, the difference of SR among four algorithms is very obvious. Although the distributions of the SR value of MS and MBPSO are more uniform than that of HLMS and CBPSOTVAC, the interquartile ranges of the former are worse than that of the latter. Moreover, outliers exist in MS and MBPSO on the SR value. In addition, we also observe the maximum, upper quartile, the mean value of HLMS is equal to or close to 1.0, 0.8, and 0.6, respectively.

Based on the above analysis, we can draw a conclusion that HLMS can obtain the best-known solution of 18 small-scale instances with a high success rate.

4.4. Comparisons on the Medium-Scale Test Set

In the second experiment, HLMS is used to solve medium-scale test instance (Test set II) to verify the performance of algorithms. The results are reported in Table 3, together with the results of other six state-of-the-art MKP algorithms, including MS, BIWOA, BMMVO, BSCA, BHHA, and BSSA. Note that these six algorithms are all novel swarm intelligence algorithms proposed in recent years and it is meaningful to select them for comparative study of MKP.

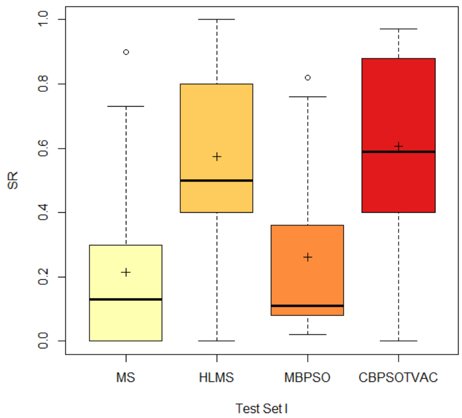

Figure 2. Boxplot for 4 comparative algorithms on *SR* for Test set I.

Table 3. The results of HLMS with 6 comparative algorithms for Test set II.

Prob.	$n \times m$	**Opt.**		**MS**	**HLMS**	**BIWOA**	**BMMVO**	**BSCA**	**BHHA**	**BSSA**
Weish01	30 × 5	4554	*Best*	**4554**	**4554**	**4554**	**4554**	**4554**	**4554**	**4554**
			Worst	4477	4534	**4554**	**4554**	4534	**4554**	**4554**
			PDev	0.38	0.18	**0.00**	**0.00**	**0.09**	**0.00**	**0.00**
Weish02	30 × 5	4536	*Best*	**4536**	**4536**	**4536**	**4536**	**4536**	**4536**	**4536**
			Worst	4440	4504	**4536**	**4536**	**4536**	**4536**	**4536**
			PDev	0.38	0.02	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
Weish03	30 × 5	4115	*Best*	4106	**4115**	4106	4106	4106	4106	4106
			Worst	**4106**	**4106**	**4106**	**4106**	**4106**	**4106**	**4106**
			PDev	0.22	0.19	3.97	3.05	**0.00**	2.24	3.97
Weish04	30 × 5	4561	*Best*	**4561**	**4561**	**4561**	**4561**	**4561**	**4561**	**4561**
			Worst	4505	4531	**4561**	**4561**	**4561**	**4561**	**4561**
			PDev	0.37	0.09	0.00	0.00	0.00	0.00	0.00
Weish05	30 × 5	4514	*Best*	**4514**	**4514**	**4514**	**4514**	**4514**	**4514**	**4514**
			Worst	**4514**	**4514**	**4514**	**4514**	**4514**	**4514**	**4514**
			PDev	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Weish06	40 × 5	5557	*Best*	**5557**	**5557**	**5557**	**5557**	**5557**	**5557**	**5557**
			Worst	5502	5542	5542	5542	5542	5544	**5557**
			PDev	0.40	0.09	0.12	0.14	0.14	0.02	0.00
Weish07	40 × 5	5567	*Best*	**5567**	**5567**	**5567**	**5567**	**5567**	**5567**	**5567**
			Worst	5360	5542	**5567**	**5567**	**5567**	**5567**	**5567**
			PDev	0.53	0.03	0.00	0.00	0.00	0.00	0.00
Weish08	40 × 5	5605	*Best*	**5605**	**5605**	**5605**	**5605**	**5605**	**5605**	**5605**
			Worst	5478	5603	**5605**	5603	5603	**5605**	**5605**
			PDev	0.30	0.01	0.00	0.03	0.01	0.00	0.00
Weish09	40 × 5	5246	*Best*	**5246**	**5246**	**5246**	**5246**	**5246**	**5246**	**5246**
			Worst	5185	**5246**	**5246**	**5246**	**5246**	**5246**	**5246**
			PDev	0.08	0.00	0.00	0.00	0.00	0.00	0.00
Weish10	50 × 5	6339	*Best*	**6339**	**6339**	6323	6303	6303	6303	6303
			Worst	6255	6280	**6303**	**6303**	**6303**	**6303**	**6303**
			PDev	0.55	0.04	0.31	0.56	0.56	0.56	0.56
Weish11	50 × 5	5643	*Best*	**5643**	**5643**	**5643**	**5643**	**5643**	**5643**	**5643**
			Worst	5592	**5643**	**5643**	**5643**	**5643**	**5643**	**5643**
			PDev	0.03	0.00	0.00	0.00	0.00	0.00	0.00
Weish12	50 × 5	6339	*Best*	**6339**	**6339**	6302	6301	6302	6302	6302
			Worst	6090	**6304**	6302	6301	6301	6301	6301
			PDev	0.94	0.07	0.58	0.59	0.59	0.59	0.59
Weish13	50 × 5	6159	*Best*	**6159**	**6159**	**6159**	**6159**	**6159**	**6159**	**6159**
			Worst	6025	6025	**6159**	**6159**	**6159**	**6159**	**6159**
			PDev	0.98	0.23	0.00	0.00	0.00	0.00	0.00

Table 3. *Cont.*

Prob.	$n \times m$	Opt.		MS	HLMS	BIWOA	BMMVO	BSCA	BHHA	BSSA
Weish14	60 × 5	6954	*Best*	**6954**	**6954**	6923	6923	6923	6923	6923
			Worst	6769	6902	6900	6900	6900	**6923**	**6923**
			PDev	0.80	0.28	0.71	0.57	0.66	0.44	0.44
Weish15	60 × 5	7486	*Best*	**7486**	**7486**	**7486**	**7486**	**7486**	**7486**	**7486**
			Worst	7199	7442	7453	7449	**7486**	**7486**	**7486**
			PDev	0.84	0.08	0.08	0.11	0.00	0.00	0.00
Weish16	60 × 5	7289	*Best*	**7289**	**7289**	**7289**	**7289**	**7289**	**7289**	**7289**
			Worst	6942	7221	**7288**	7281	7281	7281	7281
			PDev	0.96	0.14	0.01	0.10	0.09	0.10	0.10
Weish17	60 × 5	8633	*Best*	**8633**	**8633**	**8633**	8624	**8633**	**8633**	**8633**
			Worst	8141	8633	8575	8497	8506	**8633**	**8633**
			PDev	0.31	0.00	0.39	0.96	0.42	0.00	0.00
Weish18	70 × 5	9580	*Best*	9540	**9580**	9560	9456	9573	**9580**	9573
			Worst	8857	9525	9461	9318	9451	9521	**9527**
			PDev	1.42	0.11	0.65	1.92	0.62	0.27	0.17
Weish19	70 × 5	7698	*Best*	**7698**	**7698**	**7698**	**7698**	**7698**	**7698**	**7698**
			Worst	7448	7674	7632	7629	**7698**	**7698**	**7698**
			PDev	0.85	0.01	0.38	0.35	0.00	0.00	0.00
Weish20	70 × 5	9450	*Best*	**9450**	**9450**	**9450**	9445	**9450**	**9450**	**9450**
			Worst	9306	9408	9400	9365	9433	9445	**9450**
			PDev	0.49	0.03	0.23	0.57	0.02	0.01	0.00
Weish21	70 × 5	9074	*Best*	**9074**	**9074**	**9074**	**9074**	**9074**	**9074**	**9074**
			Worst	8922	9008	9016	8969	9033	**9074**	**9074**
			PDev	0.44	0.05	0.19	0.64	0.03	0.00	0.00
Weish22	80 × 5	8947	*Best*	8790	**8929**	8909	8886	8909	8912	8912
			Worst	7904	8708	8908	8886	8886	8909	**8912**
			PDev	5.51	0.58	0.43	0.68	0.63	0.39	0.39
Weish23	80 × 5	8344	*Best*	8170	**8344**	8303	8250	**8344**	**8344**	**8344**
			Worst	7246	8154	8245	8233	8245	8250	**8303**
			PDev	5.74	0.85	0.66	1.16	0.89	0.71	0.26
Weish24	80 × 5	10,220	*Best*	10,189	**10,220**	10,189	10,058	10,215	**10,202**	**10,220**
			Worst	9807	10,091	10,053	9787	10,042	**10,134**	10,132
			PDev	1.70	0.16	1.23	3.14	0.86	0.57	0.48
Weish25	80 × 5	9939	*Best*	9922	**9939**	9885	9844	**9939**	**9939**	**9939**
			Worst	9703	9885	9808	9710	9885	**9915**	**9915**
			PDev	1.02	0.03	0.94	1.63	0.20	0.21	0.11
Weish26	90 × 5	9584	*Best*	9581	**9584**	9575	9575	9575	9575	9575
			Worst	8904	9514	9477	9439	9476	9488	**9575**
			PDev	1.66	0.19	0.69	1.17	0.92	0.24	0.09
Weish27	90 × 5	9819	*Best*	9764	**9819**	9778	9589	9764	9764	9764
			Worst	9319	9764	**9773**	9487	9631	9678	9764
			PDev	2.09	0.50	0.45	2.53	0.88	0.61	0.56
Weish28	90 × 5	9492	*Best*	**9492**	**9492**	9454	9400	9454	9454	9454
			Worst	9034	**9438**	9411	9183	9400	9400	9400
			PDev	1.54	0.19	0.49	1.70	0.78	0.62	0.45
Weish29	90 × 5	9410	*Best*	9369	**9410**	9369	9369	9369	9369	9369
			Worst	8927	**9369**	**9369**	9135	**9369**	**9369**	**9369**
			PDev	1.45	0.40	0.43	1.75	0.43	0.43	0.43
Weish30	90 × 5	11,191	*Best*	11,148	**11,187**	11,121	11,025	11,169	11,169	11,169
			Worst	10,808	**11,155**	10,979	10,790	10,948	11,135	11,154
			PDev	1.44	0.08	1.23	2.49	0.61	0.27	0.20
	#*Opt*			20	**27**	16	14	18	19	19
	#*Worst*			2	9	12	10	12	18	**23**
	The mean of *PDev*			1.11	**0.15**	0.47	0.86	0.31	0.28	0.29
	p-value (*PDev*)			0.000		0.011	0.000	0.033	0.328	0.848

From Table 3, the results demonstrate that HLMS reaches the optimum solutions on 27 out of 30 instances, while the six comparative algorithms obtain the optimum solutions only on 20, 16, 14, 18, 19, and 19, respectively. In terms of #Worst, BSSA outperforms the other six algorithms on 23 instances. PDev measures the deviation between the mean and the best-known solution. The small mean value of PDev also confirms the superiority of HLMS. Comprehensively speaking, MS has the worst performance among all the comparative algorithms. Moreover, there are significant differences (p-value < 0.05) between the comparisons of the first four groups concerning PDev.

Figure 3 presents the box plots of the *PDev* values for all the comparative algorithms. The span of each box implicitly reflects the stability of the algorithm. The smaller the span is, the better the stability of the algorithm is. As can be seen from Figure 3, HLMS has a significant advantage over the other six algorithms since the span of the box for HLMS is obviously smaller than that of the other comparative algorithms. It should be noted that the dot in Figure 3. represent outliers.

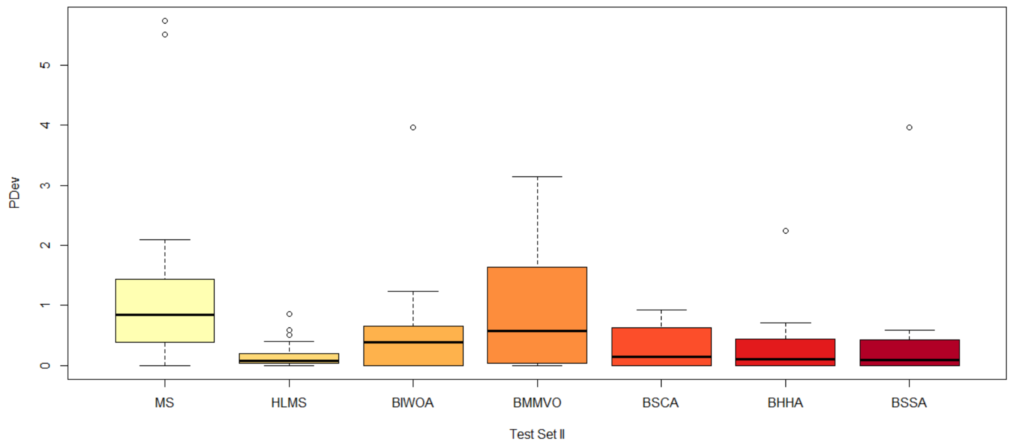

Figure 3. Boxplot for 7 comparative algorithms on *PDev* for Test set II.

In summary, the results in Table 3 and Figure 3 indicate that our HLMS algorithm is very competitive compared to the other six MKP algorithms. The findings are based on the fact that the GHS learning scheme enhances the global search ability of HLMS. On this basis, Baldwinian learning can effectively adjust the shape of search space and thereby provides good search paths towards the best solutions.

4.5. Comparisons on the Large-Scale Test Set

In the third experiment, the performance of HLMS is verified by solving large-scale problems, and the comparative results on Test set III and Test set IV are reported in Tables 4 and 5. In order to make a fair comparison with different classical algorithms using appropriate evaluation criteria, the experiment is divided into three groups on different scale instances.

4.5.1. Performance Comparison on Test Set III (cb1–cb3)

Table 4 summarizes the experimental results of the first group large-scale benchmarks. From Table 4, it can be seen clearly that the proposed HLMS still keeps the best performance in terms of all six evaluation criteria. Specifically speaking, HLMS outperforms the other comparative algorithms. In addition, the p-value indicates that there is significant difference between HLMS and MS, BGSA, BHTPSO, and BHTPSO-QI in terms of mean. However, the p-value is 0.201 for MMVO (p-value > 0.05) and then reject the null hypothesis. Hence, insignificant difference can be detected between HLMS and MMVO.

Table 4. The results of HLMS with 5 comparative algorithms for Test set III (cb1-cb3).

Prob.	$n \times m$	Opt.	Profit	MS	HLMS	MMVO	BGSA	BHTPSO	BHTPSO-QI
cb1-1	100 × 5	24,381	*Best*	24,253	**24,381**	24,192	24,152	24,169	24,301
			Mean	24,004	**24,301**	24,050	23,835	23,822	23,821
			Worst	23,311	**24,238**	23,920	23,175	23,415	23,287
cb1-2	100 × 5	24,274	*Best*	24,258	**24,274**	**24,274**	23,986	24,109	23,944
			Mean	23,934	24,231	**24,274**	23,536	23,657	23,688
			Worst	23,366	24,116	**24,274**	23,177	22,953	23,375
cb1-3	100 × 5	23,551	*Best*	23,538	**23,551**	23,538	23,386	23,435	23,418
			Mean	23,272	**23,521**	23,520	23,041	23,072	23,073
			Worst	22,953	23,468	**23,494**	22,543	22,678	22,621
cb1-4	100 × 5	23,534	*Best*	23,256	**23,503**	23,288	23,172	23,253	23,192
			Mean	23,024	**23,420**	23,120	22,863	22,928	22,923
			Worst	22,542	**23,288**	23,042	22,468	22,507	22,234
cb1-5	100 × 5	23,991	*Best*	23,845	**23,966**	23,947	23,755	23,815	23,774
			Mean	23,567	**23,937**	23,900	23,459	23,473	23,527
			Worst	23,062	**23,836**	23,799	23,106	23,155	23,053
cb2-1	250 × 5	59,312	*Best*	58,084	**59,063**	58,473	57,565	57,814	57,800
			Mean	57,369	**58,862**	58,240	56,554	56,874	56,685
			Worst	55,984	**58,653**	58,112	55,191	54,935	55,255
cb2-2	250 × 5	61,472	*Best*	60,248	**61,295**	60,692	60,057	59,982	59,767
			Mean	59,386	**61,051**	60,390	58,613	58,588	58,680
			Worst	58,167	**60,870**	60,194	57,707	56,807	56,821
cb2-3	250 × 5	62,130	*Best*	61,212	**61,767**	61,702	59,936	60,630	60,524
			Mean	59,922	**61,552**	61,330	58,975	59,234	59,186
			Worst	57,885	**61,303**	61,158	57,723	57,435	57,278
cb2-4	250 × 5	59,463	*Best*	58,386	**59,140**	58,441	57,970	57,736	57,884
			Mean	57,752	**58,922**	58,300	56,744	56,773	56,584
			Worst	56,763	**58,710**	58,163	55,371	55,589	55,164
cb2-5	250 × 5	58,951	*Best*	57,755	**58,605**	58,082	56,959	57,378	57,550
			Mean	56,929	**58,390**	58,300	55,961	56,129	56,361
			Worst	56,326	58,088	**58,163**	54,637	54,364	53,929
cb3-1	500 × 5	120,148	*Best*	116,296	119,101	**119,978**	111,206	114,493	114,438
			Mean	115,444	118,457	**119,900**	108,930	111,017	111,469
			Worst	114,634	117,842	**119,810**	106,951	106,454	107,005
cb3-2	500 × 5	117,879	*Best*	113,732	**116,227**	115,634	108,522	112,821	112,147
			Mean	112,257	**115,704**	115,400	106,631	109,276	109,247
			Worst	111,381	115,053	**115,222**	104,519	100,118	104,696
cb3-3	500 × 5	121,131	*Best*	117,666	**119,990**	119,156	111,271	114,774	116,099
			Mean	116,367	**119,468**	118,900	109,430	112,035	112,001
			Worst	115,160	**119,054**	118,651	107,683	106,406	104,627
cb3-4	500 × 5	120,804	*Best*	116,454	119,015	**119,124**	111,283	115,828	114,327
			Mean	115,396	118,386	**118,900**	109,062	112,200	111,671
			Worst	114,100	117,572	**118,623**	107,061	106,222	107,578
cb3-5	500 × 5	122,319	*Best*	117,900	120,918	**121,141**	112,391	115,889	117,242
			Mean	116,767	120,278	**120,800**	110,564	112,253	113,364
			Worst	115,062	119,519	**120,401**	108,670	102,820	103,910
#Best				0	**12**	4	0	0	0
#Mean				0	**11**	4	0	0	0
#Worst				0	**8**	7	0	0	0
AE of *Best*				1.91%	**0.57%**	0.98%	3.97%	2.70%	2.72%
AE of *Mean*				3.08%	**0.92%**	1.23%	5.63%	4.81%	4.75%
AE of *Worst*				4.81%	**1.36%**	1.48%	7.49%	8.29%	8.09%
Rank of *AE* of *Best*				3	**1**	2	6	4	5
p-value (*Mean*)				0.000		0.201	0.000	0.000	0.000

Table 5. The results of HLMS with 5 comparative algorithms for Test set III (cb4–cb5).

Prob.	$n \times m$	Opt.	Profit	MS	HLMS	BGSA	BHTPSO	BHTPSO-QI
cb4-1	100 × 10	23,064	*Best*	22,753	**23,055**	22,836	22,905	22,876
			Mean	22,459	**22,914**	22,334	22,425	22,449
			Worst	22,080	**22,753**	21,975	21,980	21,999
cb4-2	100 × 10	22,801	*Best*	22,611	**22,743**	22,441	22,573	22,408
			Mean	22,255	**22,629**	21,991	22,047	22,017
			Worst	21,622	**22,407**	21,435	21,322	21,454
cb4-3	100 × 10	22,131	*Best*	21,886	**22,131**	21,849	21,797	21,949
			Mean	21,466	**21,908**	21,313	21,342	21,461
			Worst	20,841	**21,855**	20,957	20,958	20,886
cb4-4	100 × 10	22,772	*Best*	22,319	**22,717**	22,325	22,418	22,376
			Mean	21,992	**22,528**	21,961	22,037	22,029
			Worst	21,465	22,016	21,488	21,228	21,533
cb4-5	100 × 10	22,751	*Best*	22,440	**22,751**	22,168	22,215	22,254
			Mean	22,132	**22,603**	21,840	21,822	21,903
			Worst	21,738	**22,272**	21,271	21,362	21,339
cb5-1	250 × 10	59,187	*Best*	57,757	**58,903**	56,928	57,530	57,036
			Mean	56,708	**58,477**	55,759	55,854	55,960
			Worst	55,510	**58,182**	54,217	53,570	53,381
cb5-2	250 × 10	58,781	*Best*	57,363	**58,346**	56,337	56,568	56,490
			Mean	56,793	**58,098**	55,455	55,443	55,708
			Worst	56,126	**57,792**	53,739	53,274	52,907
cb5-3	250 × 10	58,097	*Best*	56,690	**57,674**	55,573	56,426	55,982
			Mean	56,024	**57,417**	54,638	54,793	54,727
			Worst	55,281	**57,044**	53,516	52,871	52,714
cb5-4	250 × 10	61,000	*Best*	59,930	**60,505**	58,595	59,030	59,077
			Mean	58,934	**60,282**	57,766	58,057	57,721
			Worst	57,765	**59,870**	56,701	56,254	53,774
cb5-5	250 × 10	58,092	*Best*	56,863	**57,468**	56,186	56,217	56,204
			Mean	56,066	**57,220**	54,850	54,941	54,872
			Worst	55,182	**56,869**	53,612	51,850	50,832
cb6-1	500 × 10	117,821	*Best*	113,362	**116,015**	108,487	110,996	111,669
			Mean	112,541	**115,379**	105,760	107,698	108,367
			Worst	111,397	**114,509**	102,725	104,239	103,802
cb6-2	500 × 10	119,249	*Best*	115,022	**117,778**	109,569	114,262	113,001
			Mean	114,250	**117,102**	106,775	108,648	109,197
			Worst	112,596	**116,418**	103,478	100,740	100,764
cb6-3	500 × 10	119,215	*Best*	115,419	**117,345**	109,705	113,987	112,419
			Mean	114,372	**116,842**	106,853	108,576	109,004
			Worst	113,495	**116,115**	104,565	102,439	103,703
cb6-4	500 × 10	118,829	*Best*	115,038	**117,281**	108,628	112,476	112,198
			Mean	113,444	**116,446**	105,679	107,692	107,796
			Worst	112,405	**115,872**	102,679	101,860	99,470
cb6-5	500 × 10	116,530	*Best*	112,971	**114,909**	106,972	109,567	109,287
			Mean	111,707	**114,183**	104,509	106,217	106,212
			Worst	110,156	**113,533**	102,665	100,836	100,509
#Best				0	**15**	0	0	0
#Mean				0	**15**	0	0	0
#Worst				0	**14**	0	0	0
AE of *Best*				2.30%	**0.76%**	4.58%	3.25%	3.52%
AE of *Mean*				3.57%	**1.33%**	6.58%	5.92%	5.78%
AE of *Worst*				5.20%	**2.12%**	8.77%	9.64%	10.11%
Rank of *AE* of *Best*				2	**1**	5	3	4
p-value (*Mean*)				0.000		0.000	0.000	0.000

The performance comparison of six methods based on *AE* is plotted in Figure 4. It is evident that the axis of HLMS on radar charts has a point nearer to the center in comparison with the other five algorithms when considering *AE* of *Best*, *AE* of *Mean*, and *AE* of *Worst*, which indicates that it is more effective with respect to quality of solutions. It can be considered that HLMS obtained optimal or near-optimal for most of the instances in terms of *Best*, *Mean*, and *Worst*, and could beat all the other competing algorithms.

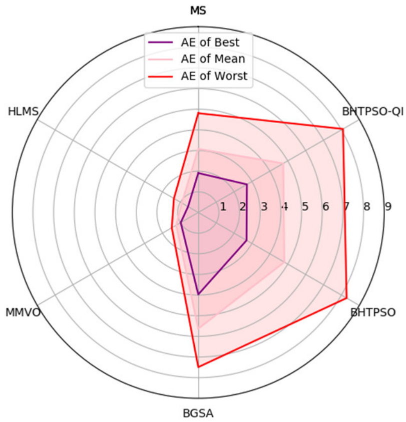

Figure 4. The performance comparison of 6 methods based on *AE* for Test set III (cb1–cb3).

To observe the stability of MS and HLMS more intuitively, the error bar based on variance and the trends plot based on *Std* are shown in Figures 5 and 6, respectively. It can be observed clearly from Figure 5 that the variances of HLMS are apparently smaller than that of MS for all benchmarks. Moreover, the variances will increase with the growth of the scale of instances. It is clear from Figure 6 that the trend lines of HLMS are located in the relatively low area for ten instances of cb1 and cb2. However, the curve of cb3 has an upward trend. In brief, the curve of MS is higher than that of HLMS, which indicates that HLMS has more stable performance than MS.

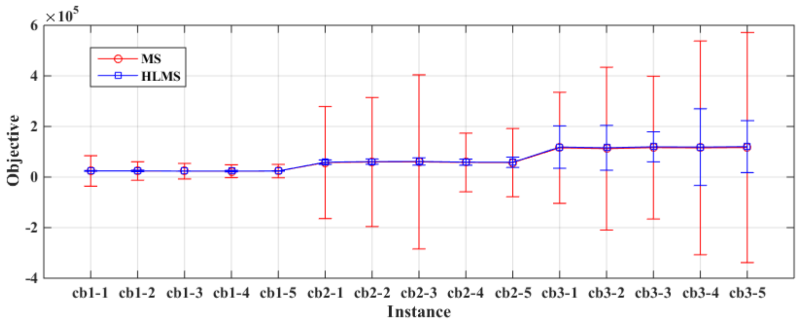

Figure 5. The error bars (Variance) for MS and HLMS on Test set III (cb1–cb3).

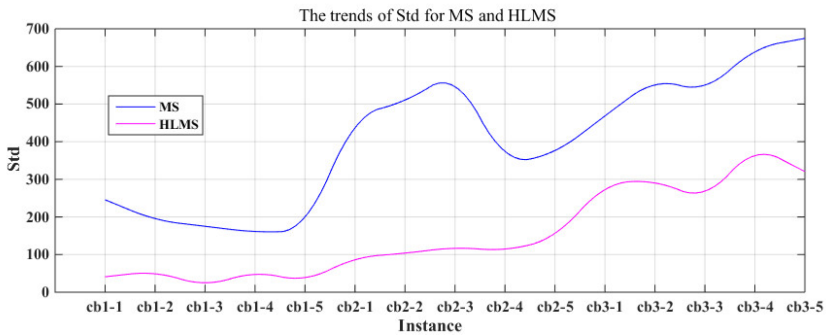

Figure 6. The trends of *Std* for MS and HLMS on Test set III (cb1–cb3).

4.5.2. Performance Comparison on Test Set III (cb4–cb6)

Table 5 summarizes the experimental results of the second group large-scale benchmarks. Table 5 shows that HLMS is also very efficient for 15 large instances with $m = 10$. Moreover, HLMS is superior to other five algorithms in absolute advantage, which is confirmed by the small p-values ($0.000 \leq 0.05$).

Similarly, radar charts are plotted to visualize three evaluation criteria, *AE* of *Best*, *AE* of *Mean*, and *AE* of *Worst* in Figure 7. From Figure 7, the phenomenon is almost consistent with Figure 4. The point on the HLMS axis is very close to the center point. By implication, HLMS has smaller *AE* value compared with other algorithms.

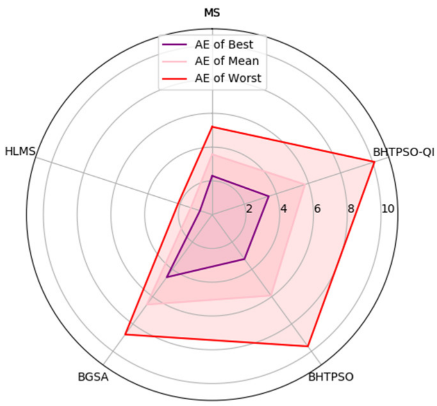

Figure 7. The performance comparison of 5 methods based on *AE* for Test set III (cb4–cb6).

The error bar based on variance for HLMS and MS is illustrated in Figure 8, which is to assess the stability of algorithms. As can be seen from Figure 8, the variance of HLMS is almost unaffected by the scale of MKP. However, with the expansion of the scale, the variance of MS is increasing gradually.

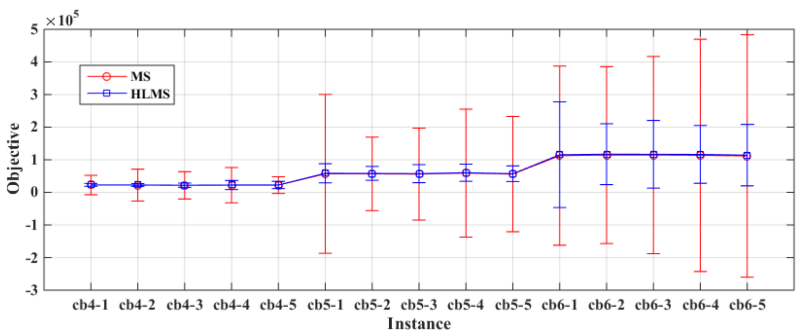

Figure 8. Error bars (Variance) for MS and HLMS on Test set III (cb4–cb6).

The trend plot of *Std* for HLMS and MS is given in Figure 9. It can be seen from Figure 9 that the trend curve of MS is significantly higher than that of HLMS, which further indicates that HLMS has better stability than MS.

4.5.3. Performance Comparison on Test Set IV

Table 6 summarizes the experimental results of the third group large-scale benchmarks. Overall, HLMS still outperforms all other comparative algorithms. In terms of #*Best*, #*Worst*, and #*Mean*, HLMS obtains a better result respectively on 5, 1, and 3 out of 9 instances. The results of BIWOA are respectively on 2, 2, and 2 out of 9 instances. BSSA with better performance obtains 2, 6, and 5 out of 9 instances, respectively. For the significance, the p-values for BMMVO and BSCA are both smaller than 0.05 concerned with *Mean*, except for MS, BIWOA, BHHA, and BSSA, which indicates that the difference between HLMS and most comparative algorithms is not significant when facing Test set IV.

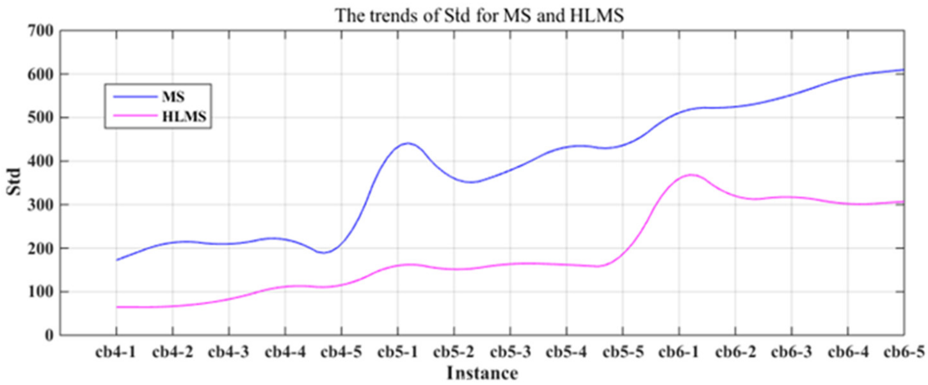

Figure 9. The trends of *Std* for MS and HLMS on Test set III (cb4–cb6).

Table 6. The results of HLMS with 6 comparative algorithms for Test set IV.

Prob.	$n \times m$	Opt.		MS	HLMS	BIWOA	BMMVO	BSCA	BHHA	BSSA
GK01	100 × 15	3766	*Best*	3732	**3752**	3743	3698	3725	3746	3744
			Worst	3714	3722	3731	3666	3690	3728	**3738**
			Mean	3721	**3742**	3736	3678	3705	3734	**3742**
GK02	100 × 25	3958	*Best*	3920	3948	**3949**	3885	3913	3929	3939
			Worst	3900	**3928**	3924	3859	3883	3915	3924
			Mean	3908	**3938**	3931	3871	3897	3919	3934
GK03	150 × 25	5656	*Best*	5585	5610	**5613**	5561	5563	5580	5606
			Worst	5545	5575	5584	5507	5533	5554	**5594**
			Mean	5564	5596	5596	5519	5543	5568	**5598**
GK04	150 × 50	5767	*Best*	5702	**5733**	5712	5651	5678	5695	5712
			Worst	5677	5658	5690	5628	5652	5672	**5696**
			Mean	5688	**5710**	5701	5638	5664	5683	5704
GK05	200 × 25	7561	*Best*	7479	**7502**	7499	7365	7411	7463	7495
			Worst	7422	7413	7476	7344	7375	7426	**7477**
			Mean	7452	7474	7485	7353	7391	7443	**7488**
GK06	200 × 50	7680	*Best*	**7617**	7611	7607	7522	7551	7578	**7617**
			Worst	7561	7568	7584	7492	7520	7562	**7598**
			Mean	7591	7592	7597	7505	7532	7569	**7611**
GK07	500 × 25	19,220	*Best*	19,066	**19,151**	19,110	18,738	18,783	19,005	19,100
			Worst	19,005	18,890	**19,093**	18,635	18,689	18,961	19,048
			Mean	19,033	19,067	**19,102**	18,668	18,734	18,983	19,087
GK08	500 × 50	18,806	*Best*	18,612	18,642	18,641	18,385	18,462	18,601	**18,646**
			Worst	18,557	18,498	18,607	18,335	18,395	18,597	**18,637**
			Mean	18,582	18,594	18,627	18,361	18,428	18,598	**18,640**
GK09	1500 × 25	58,087	*Best*	57,753	**57,886**	57,868	56,746	56,932	57,547	57,346
			Worst	57,636	56,725	**57,830**	56,519	56,624	56,699	56,615
			Mean	57,676	57,547	**57,843**	56,619	56,719	57,719	56,959
#*Best*				1	**5**	2	0	0	0	2
#*Worst*				0	1	2	0	0	0	**6**
#*Mean*				0	3	2	0	0	0	**5**
AE of *Best*				0.11%	**0.07%**	0.08%	0.24%	0.19%	0.12%	0.09%
AE of *Mean*				0.16%	0.18%	**0.11%**	0.29%	0.27%	0.17%	0.13%
AE of *Worst*				0.13%	**0.10%**	**0.10%**	0.27%	0.22%	0.13%	0.11%
Rank on *AE* of *Best*				4	**1**	2	7	6	5	3
p-value (*Mean*)				0.327		0.172	0.001	0.001	0.069	0.327

Graphically, Figure 10 shows the *AE* of *Best*, *AE* of *Mean*, and *AE* of *Worst* obtained by seven methods. In terms of *Best*, HLMS outperforms the other comparative algorithms in absolute small *AE* value. However, HLMS is slightly worse than BIWOA in terms of *AE* of *Mean*.

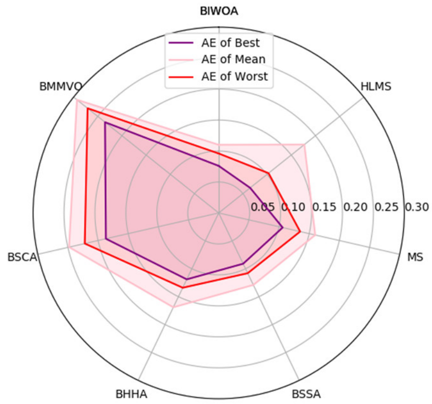

Figure 10. The performance comparison of 7 methods based on *AE* for Test set IV.

In summary, the above experimental results and comparisons show that the proposed HLMS also has excellent optimization performance in solving large-scale MKP instances, in terms of solution quality, convergence, and stability of algorithms. This is mainly due to GHS learning and Baldwinian learning being able to effectively balance exploration and exploitation in the evolution process. Compared with the original MS, the hybrid learning strategy focuses more on discovering and utilizing useful information from the whole population and whole search experience, rather than the experience of some random local individuals.

4.6. Sensitivity Analysis on the Positional Parameter and the Scale Parameter

As is known to all, Gaussian distribution and Cauchy distribution are two important distributions, which have been integrated into many algorithms and proved an effective strategy to enhance the ability of an elaborate search. The strength of Baldwinian learning in [47] is random real number obeying Gaussian distribution. However, previous studies have revealed that Cauchy mutation possesses more power in escaping local optima and converging to the global optimum. Hence, the parameters used in [47] for Gaussian distribution and the positional parameter x and the scale parameter y for Cauchy distribution are investigated in this subsection. It is noted that, to eliminate the influence of GHS, HLMS only adopted the Baldwinian learning strategy in this experiment. We tested HLMS for Gaussian distribution and different combinations for (x, y) of Cauchy distribution: (0, 1), (0, 0.5), (0, 2), and (−2, 1). Table 7 summaries the *Mean* and *SR* for 18 instances of Test set I. In addition, boxplot for five parameter combinations on *SR* is plotted in Figure 11.

From Table 7, all HLMS with four different combinations of x and y find better results than that of HLMS combined with Gaussian distribution. In terms of #*Mean*, #*SR*, and *MSR*, HLMS with four groups (x, y) shows similar results. It can be observed from Figure 11 that HLMS-C2 shows excellent comprehensive performance. HLMS-C2 has the best maximum and three-quarter quantile. Hence, considering all of the parameter combinations, we concluded that the setting $x = 0$ and $y = 0.5$ for the HLMS is an appropriate choice.

We can draw a conclusion from this experiment that it is better to use a random real obeying Cauchy distribution than Gaussian distribution as the Baldwinian learning strength. The reason may be that Cauchy mutation has stronger ability to jump from local optimum than Gaussian mutation.

4.7. The Effectiveness of the Two Components in HLMS

As mentioned above, HLMS includes two learning strategies: Baldwinian learning strategy and GHS learning strategy. The aim of this subsection is to investigate the effectiveness of these two learning strategies. Therefore, one additional experiment is conducted on Test set I and the results are summarized in Table 8. HLMS, which only adopted the Baldwinian learning strategy, is denoted as HLMS-B. HLMS, which only adopted the GHS learning strategy, is denoted as HLMS-H.

Table 7. The results of HLMS using different parameters of two distributions for Test set I.

Prob.		μ=0.5, δ=0.3	x = 0, y = 1	x = 0, y = 0.5	x = 0, y = 2	x = −2, y = 1
Sento1	*Mean*	7729	7740	7749	**7753**	7749
	SR	0.43	**0.67**	0.63	**0.67**	0.60
Sento2	*Mean*	8701	**8713**	8708	8712	**8713**
	SR	0.13	**0.33**	0.10	0.07	0.27
HP1	*Mean*	**3391**	3376	3371	3385	3367
	SR	**0.43**	0.20	0.03	0.07	0.10
HP2	*Mean*	3115	3125	3124	**3135**	3119
	SR	**0.10**	0.03	0.07	0.00	0.00
PB1	*Mean*	3045	3058	**3060**	3057	3055
	SR	0.13	0.13	**0.33**	0.17	0.20
PB2	*Mean*	3145	3145	3142	**3148**	3145
	SR	0.07	**0.13**	0.10	0.10	0.10
PB4	*Mean*	92,093	92,664	92,808	**93,229**	92,457
	SR	0.00	**0.10**	0.07	0.03	0.03
PB5	*Mean*	2098	**2118**	2112	2115	2106
	SR	0.17	0.50	0.47	**0.57**	0.37
PB6	*Mean*	753	757	757	756	**758**
	SR	0.43	**0.53**	**0.53**	**0.53**	**0.53**
PB7	*Mean*	1024	**1026**	1025	1025	**1026**
	SR	0.10	**0.13**	**0.13**	0.07	**0.13**
Weing1	*Mean*	139,551	139,615	139,397	**140,199**	138,803
	SR	**0.33**	0.07	0.23	0.20	0.13
Weing2	*Mean*	130,307	**130,370**	126,292	129,613	129,989
	SR	0.27	0.40	0.23	0.20	**0.60**
Weing3	*Mean*	92,362	**94,546**	93,373	93,033	94,489
	SR	0.27	0.50	0.50	0.30	**0.60**
Weing4	*Mean*	118,384	117,864	117,199	117,489	117,544
	SR	**0.23**	0.20	0.13	0.03	0.13
Weing5	*Mean*	96,465	95,421	**97,209**	95,372	94,898
	SR	0.43	0.33	**0.67**	0.40	0.37
Weing6	*Mean*	129,533	128,954	129,753	**129,983**	129,732
	SR	0.37	0.17	0.40	0.37	**0.47**
Weing7	*Mean*	**1,048,378**	1,038,366	1,043,595	1,035,059	1,035,752
	SR	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
Weing8	*Mean*	622,201	621,720	**622,515**	621,778	622,127
	SR	0.00	0.00	**0.03**	**0.03**	0.00
#Mean		2	5	3	**6**	3
#SR		5	**7**	6	5	6
MSR		0.22	0.25	**0.26**	0.21	**0.26**
p-value (*Mean*)			1.000	0.556	0.727	0.635

As can be seen from Table 8, compared with HLMS-B, HLMS-H, and HLMS, MS shows the worst performance in terms of *#Mean*, *#SR*, and *MSR*. The results further reveal that two learning strategies are effective in the search process. Additionally, it is noted that HLMS-B shows similar performance with MS while the performance difference between HLMS-H and MS is significant. However, the performance of HLMS integrated with two learning strategies is obviously better than that of any one.

Moreover, the convergence graphs of the average objective function values obtained by four algorithms are plotted in Figures 12 and 13 for four representative instances: Sento1, Sento2, Weing7, and Weing8. As seen from Figures 12 and 13, MS has the slowest convergence speed, while HLMS-H and HLMS have similar convergence speed and converge both faster than MS and HLMS-B.

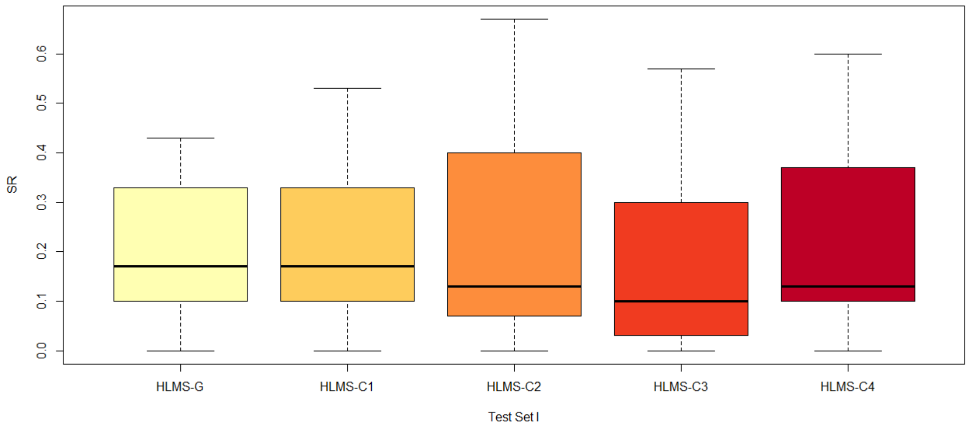

Figure 11. Boxplot for 5 parameter combinations on SR for Test set I.

Table 8. Comparisons of MS, HLMS-B, HLMS-H, and HLMS on Test set I.

Prob.		MS	HLMS-B	HLMS-H	HLMS
Sento1	*Mean*	7677	7749	**7763**	7762
	SR	0.17	0.63	0.87	**0.90**
Sento2	*Mean*	8695	8708	8717	**8719**
	SR	0.00	0.10	**0.53**	0.47
HP1	*Mean*	3370	3371	3392	**3399**
	SR	0.13	0.03	0.37	**0.40**
HP2	*Mean*	3098	3124	3135	**3158**
	SR	0.00	0.07	0.30	**0.40**
PB1	*Mean*	3036	3060	**3075**	3074
	SR	0.03	0.33	**0.50**	0.43
PB2	*Mean*	3139	3142	3172	**3178**
	SR	0.00	0.10	0.60	**0.70**
PB4	*Mean*	92,312	92,808	**93,111**	93,063
	SR	0.00	0.07	0.17	**0.30**
PB5	*Mean*	2091	2112	2117	**2125**
	SR	0.17	0.47	0.57	**0.70**
PB6	*Mean*	751	757	767	**768**
	SR	0.43	0.53	**0.83**	0.80
PB7	*Mean*	1023	1025	1028	**1030**
	SR	0.03	0.13	0.20	**0.50**
Weing1	*Mean*	141,207	139,397	141,227	**141,260**
	SR	0.90	0.23	0.83	**0.93**
Weing2	*Mean*	130,760	126,292	**130,877**	**130,877**
	SR	0.30	0.23	**0.97**	**0.97**
Weing3	*Mean*	90,866	93,373	**95,355**	94,801
	SR	0.13	0.50	**0.83**	0.80
Weing4	*Mean*	116,487	117,199	118,883	**118,956**
	SR	0.73	0.13	**0.50**	0.37
Weing5	*Mean*	95,802	97,209	98,384	**98,796**
	SR	0.23	0.67	0.87	**1.00**
Weing6	*Mean*	129,176	129,753	130,429	**130,610**
	SR	0.23	0.40	0.63	**0.97**
Weing7	*Mean*	1,069,121	1,043,595	1,073,467	**1,073,783**
	SR	**0.00**	**0.00**	**0.00**	**0.00**
Weing8	*Mean*	620,483	622,515	**622,905**	622,775
	SR	0.33	**0.03**	0.00	**0.03**
#Mean		0	0	6	**13**
#SR		1	2	7	**13**
MSR		0.21	0.26	0.53	**0.54**
p-value		0.0003	0.0003	0.155	

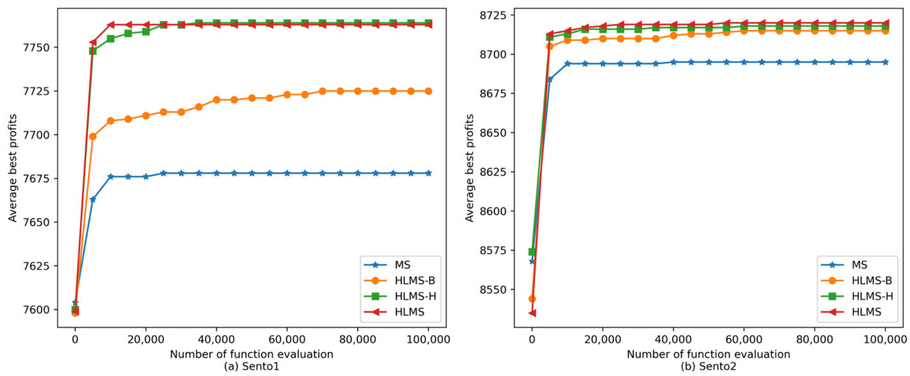

Figure 12. Convergence graphs of MS, HLMS-B, HLMS-H, and HLMS on Sento1 and Sento2.

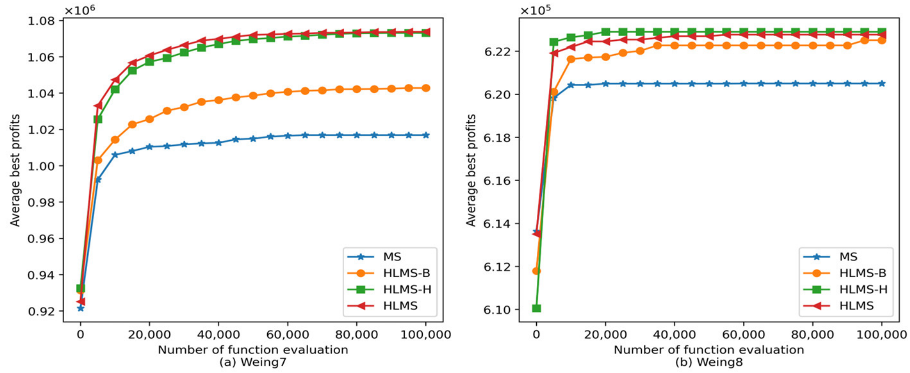

Figure 13. Convergence graphs of MS, HLMS-B, HLMS-H, and HLMS on Weing7 and Weing8.

In summary, we can draw a conclusion that the above two learning strategies can realize complementary advantages to enhance the performance of MS. Indeed, only using Baldwinian learning is not sufficient for exploitation. The conclusion is that the beneficial combination of the two strategies is significant for improving the performance of the algorithm.

4.8. Discussion

Comparison results demonstrate that the Baldwinian learning and GHS learning strategies can really improve the performance of HLMS, thereby making it better than the original MS and most other comparative algorithms on the majority of MKP instances, in terms of solution accuracy, convergence speed, and algorithm stability. The advantage of HLMS mainly lies in that GHS is used to guide the global search and dimensional learning can achieve a large jump to help solutions escape local extremum. The other reason is that Baldwinian learning as a local search strategy has the effect of changing the fitness landscape. This interaction between learning and evolution is very beneficial. Accordingly, both Baldwinian learning and GHS learning are more efficient and effective than MS alone.

In fact, based on the previous experimental results, we can find that considering small-scale MKP instances, medium-sized MKP instances, and large-scale MKP instances, the HLMS algorithm combining Baldwinian learning and GHS learning is an effective algorithm for solving MKP problems. Besides that, balancing exploration and exploitation is an important factor in metaheuristic algorithms by maintaining adequate diversity in swarm individuals so that reducing the probability of trapping in local optimal locations. In HLMS, GHS learning and Baldwinian learning respectively play the roles of exploration and exploitation, making the optimization performance of the algorithm better.

5. Conclusions and Future Work

This paper proposed a hybrid learning moth search algorithm (HLMS) inspired by the idea that the learning strategy could direct the evolutionary process. The framework proposed in this work includes two learning strategies: Baldwinian learning and GHS learning. In the search process, the two learning strategies play the role of local exploitation and global exploration, respectively.

HLMS is verified by solving the NP-hard 0–1 multidimensional knapsack problems. The experimental results on the 87 instances commonly used in literature showed that HLMS performs competitively in comparison with MS and other state-of-the-art meta-heuristics algorithms. Sensitivity analysis of Gaussian distribution and Cauchy distribution on Baldwinian learning is provided. The results proved that Cauchy mutation is more effective than Gaussian mutation as learning length. The effectiveness of two important learning strategies of HLMS is investigated. The results demonstrated that Baldwinian learning and GHS learning both play a major role in improving the performance of HLMS. MS with two learning strategies surpasses MS and MS with a single strategy. It confirms the effectiveness of our proposed learning strategies.

Future research on MS can be divided into two main directions: research on more real-world applications and research on improvements of the algorithms. Concerning the application of MS, it has the potential to solve more combinatorial optimization problems, such as maximum diversity problems (MDP), multi-objective knapsack problems (MOKP), and multi-demand multidimensional knapsack problems (MDMKP). In terms of the improvements of algorithms, more effective learning based on strategies can be adopted to enhance the search ability of the algorithm, such as orthogonal learning, reinforcement learning, and adaptive learning.

Author Contributions: Y.F. performed the methodology, investigation and wrote the draft. H.W. and Z.C. supervised the research and edited the final draft. X.L. performed the experiments. M.L. reviewed the final draft. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by National Natural Science Foundation of China (No. 61806069), Key projects of science and technology research in Colleges of Hebei Province, (No. ZD2022083), Key R & D plan project of Hebei Province (No. 22375415D), Major Projects of Guangdong Education Department for Foundation Research and Applied Research (No. 2017KZDXM081, 2018KZDXM066), Guangdong Provincial University Innovation Team Project (No. 2020KCXTD045).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Toyoda, Y. A simplified algorithm for obtaining approximate solutions to zero-one programming problems. *Manag. Sci.* **1975**, *21*, 1417–1427. [CrossRef]
2. Vasquez, M.; Vimont, Y. Improved results on the 0–1 multidimensional knapsack problem. *Eur. J. Oper. Res.* **2005**, *165*, 70–81. [CrossRef]
3. 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]
4. Weingartner, H.M.; Ness, D.N. Methods for the solution of the multidimensional 0/1 knapsack problem. *Oper. Res.* **1967**, *15*, 83–103. [CrossRef]
5. Drake, J.H.; Özcan, E.; Burke, E.K. A case study of controlling crossover in a selection hyper-heuristic framework using the multidimensional knapsack problem. *Evol. Comput.* **2016**, *24*, 113–141. [CrossRef]
6. Gilmore, P.; Gomory, R.E. The theory and computation of knapsack functions. *Oper. Res.* **1966**, *14*, 1045–1074. [CrossRef]
7. Shih, W. A branch and bound method for the multiconstraint zero-one knapsack problem. *J. Oper. Res. Soc.* **1979**, *30*, 369–378. [CrossRef]
8. Boussier, S.; Vasquez, M.; Vimont, Y.; Hanafi, S.; Michelon, P. A multi-level search strategy for the 0-1 Multidimensional Knapsack Problem. *Discret. Appl. Math.* **2010**, *158*, 97–109. [CrossRef]

9. Lai, X.; Hao, J.-K.; Glover, F.; Lü, Z. A two-phase tabu-evolutionary algorithm for the 0-1 multidimensional knapsack problem. *Inf. Sci.* **2018**, *436*, 282–301. [CrossRef]
10. Chih, M. Three pseudo-utility ratio-inspired particle swarm optimization with local search for multidimensional knapsack problem. *Swarm Evol. Comput.* **2018**, *39*, 279–296. [CrossRef]
11. Haddar, B.; Khemakhem, M.; Hanafi, S.; Wilbaut, C. A hybrid quantum particle swarm optimization for the multidimensional knapsack problem. *Eng. Appl. Artif. Intell.* **2016**, *55*, 1–13. [CrossRef]
12. Lai, X.; Hao, J.-K.; Fu, Z.-H.; Yue, D. Diversity-preserving quantum particle swarm optimization for the multidimensional knapsack problem. *Expert Syst. Appl.* **2020**, *149*, 113310. [CrossRef]
13. Wang, L.; Wang, S.-y.; Xu, Y. An effective hybrid EDA-based algorithm for solving multidimensional knapsack problem. *Expert Syst. Appl.* **2012**, *39*, 5593–5599. [CrossRef]
14. Li, Z.; Tang, L.; Liu, J. A memetic algorithm based on probability learning for solving the multidimensional knapsack problem. *IEEE Trans. Cybern.* **2020**, *54*, 2284–2299. [CrossRef] [PubMed]
15. Luo, K.; Zhao, Q. A binary grey wolf optimizer for the multidimensional knapsack problem. *Appl. Soft Comput.* **2019**, *83*, 105645. [CrossRef]
16. Zhang, B.; Pan, Q.K.; Zhang, X.L.; Duan, P.Y. An effective hybrid harmony search-based algorithm for solving multidimensional knapsack problems. *Appl. Soft Comput.* **2015**, *29*, 288–297. [CrossRef]
17. Abdel-Basset, M.; El-Shahat, D.; Faris, H.; Mirjalili, S. A binary multi-verse optimizer for 0-1 multidimensional knapsack problems with application in interactive multimedia systems. *Comput. Ind. Eng.* **2019**, *132*, 187–206. [CrossRef]
18. García, J.; Maureira, C. A KNN quantum cuckoo search algorithm applied to the multidimensional knapsack problem. *Appl. Soft Comput.* **2021**, *102*, 107077. [CrossRef]
19. Bolaji, A.L.a.; Okwonu, F.Z.; Shola, P.B.; Balogun, B.S.; Adubisi, O.D. A modified binary pigeon-inspired algorithm for solving the multi-dimensional knapsack problem. *J. Intell. Syst.* **2021**, *30*, 90–103. [CrossRef]
20. Feng, Y.; Wang, G.-G. A binary moth search algorithm based on self-learning for multidimensional knapsack problems. *Future Gener. Comput. Syst.* **2022**, *126*, 48–64. [CrossRef]
21. Gupta, S.; Su, R.; Singh, S. Diversified sine-cosine algorithm based on differential evolution for multidimensional knapsack problem. *Appl. Soft Comput.* **2022**, *130*, 109682. [CrossRef]
22. Abdel-Basset, M.; Mohamed, R.; Sallam, K.M.; Chakrabortty, R.K.; Ryan, M.J. BSMA: A novel metaheuristic algorithm for Multi-dimensional knapsack problems: Method and comprehensive analysis. *Comput. Ind. Eng.* **2021**, *159*, 107469. [CrossRef]
23. Li, W.; Yao, X.; Zhang, T.; Wang, R.; Wang, L. Hierarchy ranking method for multimodal multi-objective optimization with local pareto fronts. *IEEE Trans. Evol. Comput.* **2022**, *27*, 98–110. [CrossRef]
24. Cacchiani, V.; Iori, M.; Locatelli, A.; Martello, S. Knapsack problems-An overview of recent advances. Part II: Multiple, multidimensional, and quadratic knapsack problems. *Comput. Oper. Res.* **2022**, *143*, 105693. [CrossRef]
25. Gao, D.; Wang, G.-G.; Pedrycz, W. Solving fuzzy job-shop scheduling problem using DE algorithm improved by a selection mechanism. *IEEE Trans. Fuzzy Syst.* **2020**, *28*, 3265–3275. [CrossRef]
26. Wang, G.-G.; Gao, D.; Pedrycz, W. Solving multi-objective fuzzy job-shop scheduling problem by a hybrid adaptive differential evolution algorithm. *IEEE Trans. Ind. Inform.* **2022**, *18*, 8519–8528. [CrossRef]
27. Shadkam, E.; Bijari, M. A novel improved cuckoo optimisation algorithm for engineering optimisation. *Int. J. Artif. Intell. Soft Comput.* **2020**, *7*, 164–177. [CrossRef]
28. Parashar, S.; Senthilnath, J.; Yang, X.-S. A novel bat algorithm fuzzy classifier approach for classification problems. *Int. J. Artif. Intell. Soft Comput.* **2017**, *6*, 108–128. [CrossRef]
29. Wang, G.-G.; Deb, S.; Gandomi, A.H.; Alavi, A.H. Opposition-based krill herd algorithm with cauchy mutation and position clamping. *Neurocomputing* **2016**, *177*, 147–157. [CrossRef]
30. Wang, G.-G.; Guo, L.; Gandomi, A.H.; Hao, G.S.; Wang, H. Chaotic krill herd algorithm. *Inf. Sci.* **2014**, *274*, 17–34. [CrossRef]
31. Li, W.; Wang, G.-G.; Alavi, A.H. Learning-based elephant herding optimization algorithm for solving numerical optimization problems. *Knowl.-Based Syst.* **2020**, *195*, 105675. [CrossRef]
32. Wang, G.-G.; Deb, S.; Cui, Z. Monarch butterfly optimization. *Neural Comput. Appl.* **2019**, *31*, 1995–2014. [CrossRef]
33. Feng, Y.; Deb, S.; Wang, G.-G.; Alavi, A.H. Monarch butterfly optimization: A comprehensive review. *Expert Syst. Appl.* **2021**, *168*, 114418. [CrossRef]
34. Ma, L.; Cheng, S.; Shi, Y. Enhancing learning efficiency of brain storm optimization via orthogonal learning design. *IEEE Trans. Syst. Man Cybern. Syst.* **2020**, *51*, 6723–6742. [CrossRef]
35. Yu, H.; Li, W.; Chen, C.; Liang, J.; Gui, W.; Wang, M.; Chen, H. Dynamic Gaussian bare-bones fruit fly optimizers with abandonment mechanism: Method and analysis. *Eng. Comput.* **2020**, *38*, 743–771. [CrossRef]
36. Mirjalili, S.; Lewis, A. The whale optimization algorithm. *Adv. Eng. Softw.* **2016**, *95*, 51–67. [CrossRef]
37. Zhou, Y.; Bao, Z.; Luo, Q.; Zhang, S. A complex-valued encoding wind driven optimization for the 0-1 knapsack problem. *Appl. Intell.* **2017**, *46*, 684–702. [CrossRef]
38. Song, J.; Chen, C.; Heidari, A.A.; Liu, J.; Yu, H.; Chen, H. Performance optimization of annealing salp swarm algorithm: Frameworks and applications for engineering design. *J. Comput. Des. Eng.* **2022**, *9*, 633–669. [CrossRef]
39. Gezici, H.; Livatyalı, H. Chaotic Harris hawks optimization algorithm. *J. Comput. Des. Eng.* **2022**, *9*, 216–245. [CrossRef]

40. Chou, J.S.; Truong, D.N. A novel metaheuristic optimizer inspired by behavior of jellyfish in ocean. *Appl. Math. Comput.* **2021**, *389*, 125535. [CrossRef]
41. Wang, L.Y.; Cao, Q.J.; Zhang, Z.X.; Mirjalili, S.; Zhao, W.G. Artificial rabbits optimization: A new bio-inspired meta-heuristic algorithm for solving engineering optimization problems. *Eng. Appl. Artif. Intell.* **2022**, *114*, 31. [CrossRef]
42. Abdollahzadeh, B.; Gharehchopogh, F.S.; Khodadadi, N.; Mirjalili, S. Mountain Gazelle Optimizer: A new Nature-inspired Metaheuristic Algorithm for Global Optimization Problems. *Adv. Eng. Softw.* **2022**, *174*, 34. [CrossRef]
43. Chen, C.; Wang, X.; Yu, H.; Wang, M.; Chen, H. Dealing with multi-modality using synthesis of Moth-flame optimizer with sine cosine mechanisms. *Math. Comput. Simul.* **2021**, *188*, 291–318. [CrossRef]
44. Wang, G.-G. Moth search algorithm: A bio-inspired metaheuristic algorithm for global optimization problems. *Memetic Comput.* **2018**, *10*, 151–164. [CrossRef]
45. Li, J.; Yang, Y.H.; An, Q.; Lei, H.; Deng, Q.; Wang, G.G. Moth Search: Variants, Hybrids, and Applications. *Mathematics* **2022**, *10*, 4162. [CrossRef]
46. Elaziz, M.A.; Xiong, S.; Jayasena, K.; Li, L. Task scheduling in cloud computing based on hybrid moth search algorithm and differential evolution. *Knowl.-Based Syst.* **2019**, *169*, 39–52. [CrossRef]
47. Strumberger, I.; Sarac, M.; Markovic, D.; Bacanin, N. Moth search algorithm for drone placement problem. *Int. J. Comput.* **2018**, *3*, 75–80.
48. Strumberger, I.; Tuba, E.; Bacanin, N.; Beko, M.; Tuba, M. Hybridized moth search algorithm for constrained optimization problems. In Proceedings of the 2018 International Young Engineers Forum (YEF-ECE), Monte de Caparica, Portugal, 4 May 2018; pp. 1–5.
49. Feng, Y.; Wang, G.-G. Binary moth search algorithm for discounted {0-1} knapsack problem. *IEEE Access* **2018**, *6*, 10708–10719. [CrossRef]
50. Feng, Y.; An, H.; Gao, X. The importance of transfer function in solving set-union knapsack problem based on discrete moth search algorithm. *Mathematics* **2019**, *7*, 17. [CrossRef]
51. Feng, Y.; Yi, J.-H.; Wang, G.-G. Enhanced moth search algorithm for the set-union knapsack problems. *IEEE Access* **2019**, *7*, 173774–173785. [CrossRef]
52. Cui, Z.; Xue, F.; Cai, X.; Cao, Y.; Wang, G.-G.; Chen, J. Detection of malicious code variants based on deep learning. *IEEE Trans. Ind. Inform.* **2018**, *14*, 3187–3196. [CrossRef]
53. Zhang, J.; Pan, L.; Han, Q.-L.; Chen, C.; Wen, S.; Xiang, Y. Deep Learning Based Attack Detection for Cyber-Physical System Cybersecurity: A Survey. *IEEE/CAA J. Autom. Sin.* **2021**, *9*, 377–391. [CrossRef]
54. Wang, F.; Wang, X.; Sun, S. A reinforcement learning level-based particle swarm optimization algorithm for large-scale optimization. *Inf. Sci.* **2022**, *602*, 298–312. [CrossRef]
55. Wang, L.; Pan, Z.; Wang, J. A review of reinforcement learning based intelligent optimization for manufacturing scheduling. *Complex Syst. Model. Simul.* **2021**, *1*, 257–270. [CrossRef]
56. Zhou, T.; Chen, M.; Zou, J. Reinforcement learning based data fusion method for multi-sensors. *IEEE/CAA J. Autom. Sin.* **2020**, *7*, 1489–1497. [CrossRef]
57. Zheng, R.-z.; Zhang, Y.; Yang, K. A transfer learning-based particle swarm optimization algorithm for travelling salesman problem. *J. Comput. Des. Eng.* **2022**, *9*, 933–948. [CrossRef]
58. Bingjie, X.; Deming, L. Q-Learning-based teaching-learning optimization for distributed two-stage hybrid flow shop scheduling with fuzzy processing time. *Complex Syst. Model. Simul.* **2022**, *2*, 113–129. [CrossRef]
59. Wang, G.-G.; Tan, Y. Improving metaheuristic algorithms with information feedback models. *IEEE Trans. Cybern.* **2019**, *49*, 542–555. [CrossRef]
60. Chen, C.; Wang, X.; Yu, H.; Zhao, N.; Wang, M.; Chen, H.; Kumarappan, N. An Enhanced Comprehensive Learning Particle Swarm Optimizer with the Elite-Based Dominance Scheme. *Complexity* **2020**, *2020*, 1–24. [CrossRef]
61. Gong, M.; Jiao, L.; Zhang, L. Baldwinian learning in clonal selection algorithm for optimization. *Inf. Sci.* **2010**, *180*, 1218–1236. [CrossRef]
62. Hinton, G.E.; Nowlan, S.J. How learning can guide evolution. In *Adaptive Individuals in Evolving Populations: Models and Algorithms*; Addison-Wesley Longman Publishing Co., Inc.: Boston, MA, USA, 1996; Volume 26, pp. 447–454.
63. Qi, Y.; Liu, F.; Liu, M.; Gong, M.; Jiao, L. Multi-objective immune algorithm with Baldwinian learning. *Appl. Soft Comput.* **2012**, *12*, 2654–2674. [CrossRef]
64. Zhang, C.; Chen, J.; Xin, B. Distributed memetic differential evolution with the synergy of Lamarckian and Baldwinian learning. *Appl. Soft Comput.* **2013**, *13*, 2947–2959. [CrossRef]
65. Geem, Z.W.; Kim, J.H.; Loganathan, G.V. A new heuristic optimization algorithm: Harmony search. *Simulation* **2001**, *76*, 60–68. [CrossRef]
66. Omran, M.G.H.; Mahdavi, M. Global-best harmony search. *Appl. Math. Comput.* **2008**, *198*, 643–656. [CrossRef]
67. 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*, 1–12. [CrossRef]
68. Keshtegar, B.; Sadeq, M.O. Gaussian global-best harmony search algorithm for optimization problems. *Soft Comput.* **2017**, *21*, 7337–7349. [CrossRef]
69. El-Abd, M. An improved global-best harmony search algorithm. *Appl. Math. Comput.* **2013**, *222*, 94–106. [CrossRef]

70. Peng, Y.; Lu, B.-L. Hybrid learning clonal selection algorithm. *Inf. Sci.* **2015**, *296*, 128–146. [CrossRef]
71. Lan, K.-T.; Lan, C.-H. Notes on the distinction of Gaussian and Cauchy mutations. In Proceedings of the 2008 Eighth International Conference on Intelligent Systems Design and Applications, Kaohsuing, Taiwan, 26–28 November 2008; pp. 272–277.
72. Mahdavi, M.; Fesanghary, M.; Damangir, E. An improved harmony search algorithm for solving optimization problems. *Appl. Math. Comput.* **2007**, *188*, 1567–1579. [CrossRef]
73. Xiao, S.; Wang, W.; Wang, H.; Tan, D.; Wang, Y.; Yu, X.; Wu, R. An Improved Artificial Bee Colony Algorithm Based on Elite Strategy and Dimension Learning. *Mathematics* **2019**, *7*, 289. [CrossRef]
74. Chu, P.C.; Beasley, J.E. A genetic algorithm for the multidimensional knapsack problem. *J. Heuristics* **1998**, *4*, 63–86. [CrossRef]
75. Puchinger, J.; Raidl, G.R.; Pferschy, U. The multidimensional knapsack problem: Structure and algorithms. *Inf. J. Comput.* **2010**, *22*, 250–265. [CrossRef]
76. Senju, S.; Toyoda, Y. An approach to linear programming with 0-1 variables. *Manag. Sci.* **1968**, *15*, B196–B207. [CrossRef]
77. Freville, A. Hard 0-1 multiknapsack test problems for size reduction methods. *Investig. Oper.* **1990**, *1*, 251–270.
78. Bansal, J.C.; Deep, K. A modified binary particle swarm optimization for knapsack problems. *Appl. Math. Comput.* **2012**, *218*, 11042–11061. [CrossRef]
79. Chih, M.; Lin, C.-J.; Chern, M.-S.; Ou, T.-Y. Particle swarm optimization with time-varying acceleration coefficients for the multidimensional knapsack problem. *Appl. Math. Model.* **2014**, *38*, 1338–1350. [CrossRef]
80. Dina, E.-G.; Badr, A.; Abd El Azeim, M. New binary particle swarm optimization with immunity-clonal algorithm. *J. Comput. Sci.* **2013**, *9*, 1534.
81. Beheshti, Z.; Shamsuddin, S.M.; Hasan, S. Memetic binary particle swarm optimization for discrete optimization problems. *Inf. Sci.* **2015**, *299*, 58–84. [CrossRef]
82. Kennedy, J.; Eberhart, R.C. A discrete binary version of the particle swarm algorithm. In Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics, Orlando, FL, USA, 12–15 October 1997; pp. 4104–4108.
83. Aaha, B.; Sm, C.; Hf, D.; Ia, D.; Mm, E.; Hc, F. Harris hawks optimization: Algorithm and applications. *Future Gener. Comput. Syst.* **2019**, *97*, 849–872.
84. Mirjalili, S.; Gandomi, A.H.; Mirjalili, S.Z.; Saremi, S.; Faris, H.; Mirjalili, S.M. Salp swarm algorithm: A bio-inspired optimizer for engineering design problems. *Adv. Eng. Softw.* **2017**, *114*, 163–191. [CrossRef]
85. 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.* **2017**, *10*, 1–20. [CrossRef]
86. Pinto, H.; Pea, A.; Valenzuela, M.; Fernández, A. A binary sine-cosine algorithm applied to the knapsack problem. In Proceedings of the Computer Science Online Conference, Zlin, Czech Republic, 24–27 April 2019; pp. 128–138.