mathematics

MDPI

Article

The Importance of Transfer Function in Solving Set-Union Knapsack Problem Based on Discrete Moth Search Algorithm

Yanhong Feng [1,2,3], Haizhong An [1,2,*] and Xiangyun Gao [1,2]

1 School of Economics and Management, China University of Geosciences, Beijing 100083, China; qinfyh@hgu.edu.cn (Y.F.); gaoxy@cugb.edu.cn (X.G.)
2 Key Laboratory of Carrying Capacity Assessment for Resource and Environment, Ministry of Natural Resources, Beijing 100083, China
3 School of Information Engineering, Hebei GEO University, Shijiazhuang 050031, China
* Correspondence: ahz369@cugb.edu.cn

Received: 1 September 2018; Accepted: 10 December 2018; Published: 24 December 2018

Abstract: Moth search (MS) algorithm, originally proposed to solve continuous optimization problems, is a novel bio-inspired metaheuristic algorithm. At present, there seems to be little concern about using MS to solve discrete optimization problems. One of the most common and efficient ways to discretize MS is to use a transfer function, which is in charge of mapping a continuous search space to a discrete search space. In this paper, twelve transfer functions divided into three families, S-shaped (named S1, S2, S3, and S4), V-shaped (named V1, V2, V3, and V4), and other shapes (named O1, O2, O3, and O4), are combined with MS, and then twelve discrete versions MS algorithms are proposed for solving set-union knapsack problem (SUKP). Three groups of fifteen SUKP instances are employed to evaluate the importance of these transfer functions. The results show that O4 is the best transfer function when combined with MS to solve SUKP. Meanwhile, the importance of the transfer function in terms of improving the quality of solutions and convergence rate is demonstrated as well.

Keywords: set-union knapsack problem; moth search algorithm; transfer function; discrete algorithm

1. Introduction

The knapsack problem (KP) [1] is still considered as one of the most challenging and interesting classical combinatorial optimization problems, because it is non-deterministic polynomial hard problem and has many important applications in reality. As an extension of the standard 0–1 knapsack problem (0–1 KP) [2], the set-union knapsack problem (SUKP) [3] is a novel KP model recently introduced in [4,5]. The SUKP finds many practical applications such as financial decision making [4], data stream compression [6], flexible manufacturing machine [3], and public key prototype [7].

The classical 0–1 KP is one of the simplest KP model in which each item has a unique value and weight. However, SUKP is constructed of a set of items $S = \{U_1, U_2, U_3, \ldots, U_m\}$ and a set of elements $U = \{u_1, u_2, u_3, \ldots, u_n\}$. Each item is associated with a subset of elements. In SUKP, each item has a nonnegative profit and each element has a nonnegative weight. The goal is to maximize the total profit of a subset of items $S* \subset S$ such that the total weight of the corresponding element does not exceed the maximum capacity of knapsack C. Hence, SUKP is more complicated and more difficult to handle than the standard 0–1 KP. Thus far, only a few researchers have studied this issue despite its practical importance and NP-hard character. For example, Goldschmidt et al. applied the dynamic programming (DP) algorithm for SUKP [3]. However, when the exact algorithm is used, no satisfactory

approximate solution is usually obtained in polynomial time. Afterwards, Ashwin [4] proposed an approximation algorithm A-SUKP for SUKP. Obviously, A-SUKP also has to face the inevitable problem, that is, how to compromise between achieving a high-quality solution and exponential runtime. Recently, He et al. [5] presented a binary artificial bee colony algorithm (BABC) to solve SUKP and comparative studies were conducted among BABC, A-SUKP, and binary differential evolution (DE) [8]. The results verified that BABC outperformed A-SUKP method. Ozsoydan et al. [9] proposed a swarm intelligence-based algorithm for the SUKP and designed an effective mutation procedure. Although this method does not require transfer functions, it lacks generality. Therefore, it is urgent to find an efficient metaheuristic algorithm to address SUKP whether from the perspective of academic research or practical application.

As a relatively novel nature-inspired metaheuristic algorithm, moth search (MS) algorithm was recently developed for continuous optimization by Wang [10]. Computational experiments have shown that MS is not only effective but also efficient when addressing unconstrained continuous optimization problems, compared with five state-of-the-art metaheuristic algorithms. Because of its relative novelty, extensive research on MS is relatively scarce, especially discrete version MS algorithm. Feng et al. presented a binary moth search algorithm (BMS) for discounted {0–1} knapsack problem (DKP) [11].

As we all know, the metaheuristic algorithm is usually discretized in two ways: direct discretization and indirect discretization. Direct discretization is usually achieved by modifying the evolutionary operator of the original algorithm to solve a particular discrete problem. This method depends on the algorithm used and the problem solved. Obviously, the disadvantages of direct discretization are lack of versatility and complicated operation. The latter is discretized by establishing a mapping relationship between continuous space and discrete space. Concretely speaking, indirect discretization is usually achieved by an appropriate transfer function to convert real-valued variables into discrete variables. Many discrete versions of swarm intelligence algorithms using transfer functions have been proposed to solve various optimization problems. Discrete binary particle swarm optimization [12], discrete firefly algorithm [13], and binary harmony search algorithm [14] are among the most typical algorithms. Through analyzing the literature, many kinds of transfer functions can be used, such as sigmoid function [12], tanh function [15], etc. However, most existing metaheuristics only consider one transfer function. Little research concentrates on the importance of transfer functions in solving discrete problems. In addition, a few studies [16,17] investigate the efficiency of multiple transfer functions.

In this paper, twelve principal transfer functions are used and then twelve new discrete MS algorithms are proposed to solve SUKP. These functions include four S-shaped transfer functions [16,17], named S1, S2, S3, and S4, respectively; four V-shaped transfer functions [16,17], named V1, V2, V3, and V4, respectively; and four other shapes transfer functions (Angle modulation method [18,19], Nearest integer method [20,21], Normalization method [22], and Rectified linear unit method [23]), named O1, O2, O3, and O4, respectively. Therefore, combining twelve transfer functions with MS algorithm, twelve discrete MS algorithms are naturally proposed, named as MSS1, MSS2, MSS3, MSS4, MSV1, MSV2, MSV3, MSV4, MSO1, MSO2, MSO3, and MSO4, respectively.

The remainder of the paper is organized as follows. In Section 2, we briefly introduce the SUKP problem and MS algorithm. The families of transfer functions and repair optimization mechanism are presented in Section 3. In Section 4, the twelve discrete MS algorithms are compared to shed light on how the transfer functions affect the performance of the algorithm. After that, the best algorithm (MSO4) is compared with five state-of-the-art methods on fifteen SUKP instances. Finally, we draw conclusions and suggest some directions for future research.

2. Background

To describe discrete MS algorithm for the SUKP, we first explain the mathematical model of SUKP and then introduce the MS algorithm.

2.1. Set-Union Knapsack Problem

The set-union knapsack problem (SUKP) [3,4] is a variant of the classical 0–1 knapsack problem (0–1 KP). More formally, the SUKP can be defined as follows: given a set of elements $U = \{u_1, u_2, u_3, \ldots, u_n\}$ and a set of items $S = \{U_1, U_2, U_3, \ldots, U_m\}$, such that S is the cover of U, and $U_i \neq \varnothing \wedge U_i \subset U$ ($i = 1, 2, 3, \ldots, m$) and each item U_i has a value $p_i > 0$. Each element u_j ($j = 1, 2, 3, \ldots, n$) has a weight $w_j > 0$. Suppose that set A consists of some items packed into the knapsack with capacity C, namely $A \subseteq S$. Then, the profit of A is defined as $P(A) = \sum_{U_i \in A} p_i$ and the weight of A is defined as $W(A) = \sum_{u_j \in \bigcup_{U_i \in A} U_i} w_j$. The objective of the SUKP is to find a subset A that maximizes the total value $P(A)$ on condition that the total weight $W(A) \leq C$. Then, the mathematical model of SUKP can be formulated as follows:

$$Max \;\; P(A) = \sum_{U_i \in A} p_i \tag{1}$$

$$subject\ to\ W(A) = \sum_{u_j \in \bigcup_{U_i \in A} U_i} w_j \leq C, \;\; A \subseteq S \tag{2}$$

where p_i ($i = 1, 2, 3, \ldots, m$), w_j ($j = 1, 2, 3, \ldots, n$), and C are all positive integers.

Recently, an integer programming model is proposed by He et al. [5] to solve SUKP easily by using metaheuristic algorithm; the new mathematical model of SUKP can be defined as follows:

$$Max \;\; f(Y) = \sum_{i=1}^{m} y_i p_i \tag{3}$$

$$subject\ to\ W(A_Y) = \sum_{j \in \bigcup_{U_i \in A_Y} U_i} w_j \leq C \tag{4}$$

Obviously, all the 0–1 vectors $Y = [y_1, y_2, y_3, \ldots, y_m] \in \{0,1\}^m$ are the potential solutions of SUKP. A solution satisfying the constraint of Equation (4) is a feasible solution; otherwise, it is an infeasible solution. $A_Y = \{U_i \mid y_i \in Y, y_i = 1, 1 \leq i \leq m\} \subseteq S$. Then, $y_i = 1$ if and only if $U_i \in A_Y$.

2.2. Moth Search Algorithm

The MS algorithm [10] is a novel metaheuristic algorithm that was inspired by the phototaxis and Lévy flights of the moths in nature, which are the two most representative characteristics of moths. The MS is akin to other population-based swarm intelligence algorithms. However, MS differs from most the population-based metaheuristic algorithms, such as genetic algorithm (GA) [24,25] and particle swarm optimization algorithm (PSO) [26,27], which consist of only one population, as, in MS, the whole population is divided into two subpopulations according to the fitness, namely subpopulation1 and subpopulation2.

The MS starts its evolutionary process by first randomly generating n moth individuals. Each moth individual represents a candidate solution to the corresponding problem with a specific fitness function. In MS, two operators are considered including Lévy flights operator and straight flight operator. Correspondingly, an individual update in subpopulation1 and subpopulation2 is generated by performing Lévy flights operator and straight flight operator, respectively.

i. Lévy flights: For each individual i in subpopulation1, it will fly around the best one in the form of Lévy flights. The resulting new solution is calculated based on Equations (5)–(7).

$$x_i^{t+1} = x_i^t + \alpha L(s) \tag{5}$$

$$\alpha = S_{max}/t^2 \tag{6}$$

$$L(s) = \frac{(\beta - 1)\Gamma(\beta - 1) sin(\frac{\pi(\beta - 1)}{2})}{\pi s^{\beta}} \tag{7}$$

where x_i^t and x_i^{t+1} denote the position of moth i at generation t and $t+1$, respectively. α denotes the scale factor related to specific problem. S_{max} is the max walk step and it takes the value 1.0 in this paper. $L(s)$ represents the step drawn from Lévy flights and $\Gamma(x)$ is the gamma function. In this paper, $\beta = 1.5$ and s can be regarded as the position of moth individual in the solution space then s^β is the β power of s.

ii. Straight flights: for each individual i in subpopulation2, it will fly towards that source of light in line. The resulting new solution is formulated as Equation (8).

$$x_i^{t+1} = \begin{cases} \lambda \times (x_i^t + \varphi \times (x_{best}^t - x_i^t)) & if\ rand > 0.5 \\ \lambda \times (x_i^t + \frac{1}{\varphi} \times (x_{best}^t - x_i^t)) & else \end{cases} \quad (8)$$

where λ and φ represent scale factor and acceleration factor, respectively. ${x^t}_{best}$ is the best individual at generation t. *Rand* is a function generating a random number uniformly distributed in (0, 1).

3. Discrete MS Optimization Method for SUKP

In this section, we describe the newly proposed discrete MS for SUKP. The main purpose of extending MS algorithm to solve the novel SUKP is to investigate the significant role of the transfer functions in terms of improving the quality of solutions and convergence rate. The basic MS algorithm was initially proposed for continuous optimization problems, while SUKP belongs to a discrete optimization problem with constraints. Therefore, the SUKP problem must contain three key elements, namely, discretization method, solution representation, and constraint handling. The three key elements are described in detail subsequently.

3.1. Transfer Functions

Transfer function is a major contributor of the discrete MS algorithm; therefore, it deserves special attention and research. In this section, 12 transfer functions are introduced. According to the shape of transfer function curve, we divide the twelve transfer functions into three groups: S-shaped transfer functions [12], V-shaped transfer functions [15], and other-shaped (O-shaped) transfer functions [19,21]. As described above, each group consists of four functions, which are named as Si, Vi, and Oi ($i = 1, 2, 3, 4$), respectively. These transfer functions are presented in Table 1 and Figure 1.

Table 1. Twelve transfer functions.

Number	Mathematical Formula
S1 [17]	$T(x) = \frac{1}{1+e^{-2x}}$
S2 [12]	$T(x) = \frac{1}{1+e^{-x}}$
S3 [17]	$T(x) = \frac{1}{1+e^{-x/2}}$
S4 [17]	$T(x) = \frac{1}{1+e^{-x/3}}$
V1 [20]	$T(x) = \left\lvert erf(\frac{\sqrt{\pi}}{2}x)\right\rvert = \left\lvert \frac{\sqrt{2}}{\pi}\int_0^{\frac{\sqrt{\pi}}{2}x} e^{-t^2}dt\right\rvert$
V2 [12]	$T(x) = \lvert\tanh(x)\rvert$
V3 [17]	$T(x) = \left\lvert \frac{x}{\sqrt{1+x^2}}\right\rvert$
V4 [17]	$T(x) = \left\lvert \frac{2}{\pi}\arctan(\frac{\pi}{2}x)\right\rvert$
O1 [18]	$T(x) = \sin(2\pi(x-a)*b*\cos(2\pi(x-a)*c)) + d$ $(a = 0, b = 1, c = 1, d = 0)$
O2 [20]	$T(x) = \lfloor\lvert x \bmod 2\rvert\rfloor$
O3 [22]	$T(x) = \frac{(x+x_{min})}{(\lvert x_{min}\rvert + x_{max})}$ $(x_{min} \le x \le x_{max})$
O4 [23]	$T(x) = x$

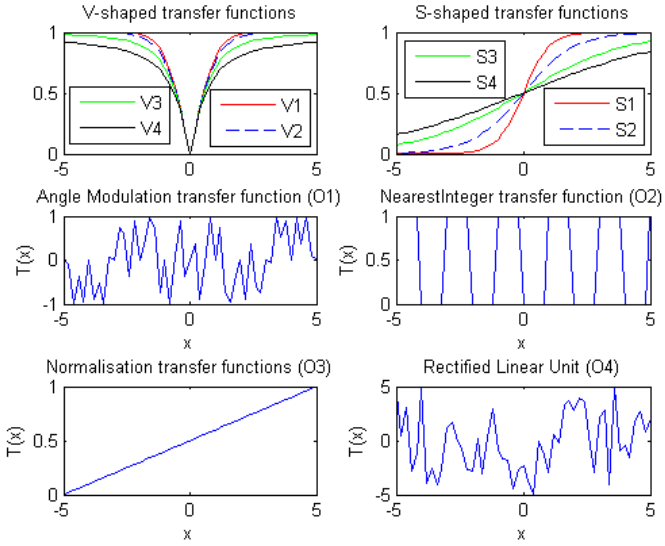

Figure 1. Twelve transfer functions.

As stated in the literature [16,17], the transfer functions define the probability that the element of position vector of each moth individual changes from 0 to 1, and vice versa. Therefore, an appropriate transfer function should ensure that a real-valued vector in a continuous search space is mapped to the value 1 in a binary search space with greater probability. Suppose applying the transfer function $T(x)$ will return a function value y ($y = 1$ or $y = 0$) through a mapping method. The probability of a transfer function with a value of 1 (PR) is displayed in Figure 2. Three groups of items, namely, 100 items, 300 items, and 500 items, were selected to count the PR value:

$$PR = \frac{\sum_{i=1}^{N} \{y_i | y_i = 1\}}{N} \times 100\% \tag{9}$$

where N represents the number of items. The value of longitudinal axis in Figure 2 is the average of PR among 100 independent runs.

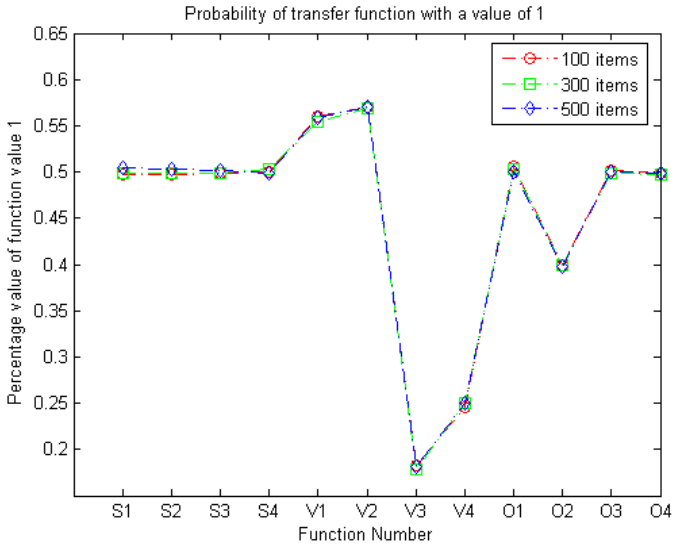

Figure 2. Probability of transfer function with a value of 1.

As shown in Figure 2, the four S-shaped transfer functions have similar PR values, which are close to 0.5. However, the PR values of the four V-shaped transfer functions differ considerably. V2 has

the best *PR* value while the *PR* value of V3 is less than 0.2. It seems that V3 combining with MS should show poor performance. Similarly, V4 also demonstrates unsatisfactory performance, with a *PR* value of less than 0.25. Of the four other shapes of transfer functions, O1, O3, and O4 obtain a similar *PR* value, that is, close to 0.5. The *PR* value of O2 is slightly smaller than that of O1, O3, and O4. In sum, according to the preliminary analysis of *PR* values, it seems that V3, V4, and O2 are not suitable for combining with MS to solve binary optimization problems.

3.2. Solution Representation

The basic MS is a real-valued algorithm and each moth individual is represented as a real-valued vector. Two main operators are defined in continuous space. However, SUKP is a discrete optimization problem with constraints and the solution is a binary vector. In this paper, the most general and simplest method, mapping the real-valued vectors into binary ones by transfer functions, is opted. Concretely speaking, a real-valued vector $X = [x_1, x_2, \ldots, x_m] \in [-a, a]^m$ still evolves in continuous space. Here, m is the number of items and a is a positive real value, and $a = 5.0$ in this paper. Then, transfer function $T(x)$ is used to map X into a binary vector $Y = [y_1, y_2, \ldots, y_m] \in \{0, 1\}^m$. According to the feature of these transfer functions, three mapping methods are as follows.

The first mapping method: Choose a transfer function from S1–S4, V1–V4, and O3.

$$y_i = \begin{cases} 1 & if\ rand() \geq T(x_i) \\ 0 & else \end{cases} \tag{10}$$

where *rand*( ) is a random number in (0, 1). In Figure 1, it can be observed that S-shaped transfer functions, V-shaped transfer functions, and O3 will return a random real number between 0 and 1. Therefore, the comparison of *rand*( ) to $T(x_i)$ equals 1 or 0. Then, the mapping procedure is shown as Table 2.

Table 2. The first mapping procedure according to *S*2 transfer function.

Element	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
X	2.96	3.32	−3.25	2.65	2.61	−1.57	−0.07	0.91	1.04	1.68
T(X)	0.95	0.97	0.04	0.93	0.93	0.17	0.48	0.71	0.74	0.84
Rand()	0.61	0.17	0.07	0.15	0.08	0.86	0.72	0.39	0.80	0.62
Y	0	0	1	0	0	1	1	0	1	0

The second mapping method: Choose the transfer function O2.

$$y_i = T(x_i) \tag{11}$$

The third mapping method: Choose either O1 or O4 as the transfer function.

$$y_i = \begin{cases} 1 & if\ T(x_i) \geq 0 \\ 0 & else \end{cases} \tag{12}$$

Then, the quality of any feasible solution Y is evaluated by the objective function f of the SUKP. Given a potential solution $Y = [y_1, y_2, \ldots, y_m]$, the objective function value f(Y) is defined by

$$f(Y) = \sum_{i=1}^{m} y_i p_i \tag{13}$$

3.3. Repair Mechanism and Greedy Optimization

Clearly, SUKP is a kind of important combinatorial optimization problem with constraints. Due to the existence of constraints, the feasible search space of decision variables becomes irregular,

which will increase the difficulty of finding the optimal solution. Among the many constraint processing techniques, repairing the infeasible solution is a common method to solve the combinatorial optimization problem. Michalewicz [28] introduced an evolutionary system based on repair technology. Obviously, repairing technique is dependent on specific problems and different repairing method must be designed for different problems. Consequently, He et al. [5] designed a repairing and optimization algorithm (named S-GROA) for SUKP, which can not only repair infeasible solutions but also further optimize feasible solutions. On the basis of S-GROA [5], a quadratic greedy repair and optimization strategy (QGROS) is proposed by Liu et al. [29]. In this paper, QGROS is adopted. The preprocessing phase of QGROS can be summarized as follows:

(1) Compute the frequency d_j of the element j (j = 1, 2, 3, ... , n) in the subsets U_1, U_2, U_3, ... , U_m.

(2) Calculate the unit weight R_i of the item i (i = 1, 2, 3, ... , m).

$$R_i = \sum_{j \in U_i} (w_j / d_j) \tag{14}$$

(3) Record the profit density of each item in S according to PD_i.

$$PD_i = p_i / R_i (i = 1,\ 2,\ 3, \ldots , m) \tag{15}$$

(4) Sort all the items in a non-ascending order based on PD_i (i = 1, 2, 3, ..., m) and then the index value recorded in an array H[1 ... m].

(5) Define a term $A_{\mathrm{Y}} = \{U_i | y_i \in \mathrm{Y} \wedge y_i = 1, 1 \leq i \leq m\}$ for any binary vector Y = [y_1, y_2, ... , y_m] $\in$ {0, 1}^{m}.

The pseudocode of QGROS [29] is outlined in Algorithm 1.

Algorithm 1. QGROS algorithm for SUKP.

Begin

Step 1: Input: the candidate solution $\mathrm{Y} = [y_1, y_2, \ldots, y_m] \in \{0,1\}^m$, H[1 ... m].

Step 2: Initialization. The m-dimensional binary vector Z = [0, 0, ... , 0].

Step 3: Greedy repair stage

For i = 1 **to** m **do**

If $(y_{\mathrm{H}[i]} = 1 \ and \ W(A_Z \cup \{\mathrm{H}[i]\}) \leq C)$

$Z_{\mathrm{H}[i]} = 1 \ and \ A_Z = A_Z \cup \{\mathrm{H}[i]\}$.

End if

End for

Y $\leftarrow$ Z.

Step 4: Quadratic greedy stage

Do not consider the elements that have been packed into the knapsack,

recalculate d_j (j = 1, 2, 3, ... , n), R_i (i = 1, 2, 3, ... , m), and H[1 ... m]

Step 5: Optimization sate

For i = 1 **to** m **do**

If $(y_{\mathrm{H}[i]} = 0 \ and \ W(A_Z \cup \{\mathrm{H}[i]\}) \leq C)$

$y_{\mathrm{H}[i]} = 1 \ and \ A_Y = A_Y \cup \{\mathrm{H}[i]\}$.

End if

End for

Step 6: Output: $\mathrm{Y} = [y_1, y_2, \ldots, y_m] \ and \ f(\mathrm{Y})$

End.

In Algorithm 1, we can observe that QGROS consists of three stages. The first stage is determining whether the constraints are met for the items in the potential solution that are ready to be packed into the knapsack. At this stage, items in the potential solution that are intended to be packed into

knapsack but violate constraints will be removed. Therefore, all solutions are feasible after this stage. The second stage is recalculating the frequency of each element, the unit weight of each item, and the array H[1 ... m]. The third stage is optimizing the remaining items by loading appropriate items into the knapsack with the aim of maximizing the use of the remaining capacity. At this stage, items in the feasible solution that are not intended to be loaded in the knapsack but satisfy the constraints will be loaded. Hence, after this stage, all solutions remain feasible and the quality of solutions is improved.

3.4. The Main Scheme of Discrete MS for SUKP

Having discussed all the components of the discrete MS algorithms in detail, the complete procedure is outlined in Algorithm 2.

3.5. Computational Complexity of the Discrete MS Algorithm

Computational complexity is the main criterion for evaluating the running time of an algorithm, which can be calculated according to its structure and implementation. In Algorithm 2, it can be seen that the computing time in each iteration is mainly dependent on the number of moths, problem dimension, and sorting of items as well as moth individual in each iteration. In addition, the computational complexity is mainly determined by Steps 1–4. In Step 1, since the Quicksort algorithm is used, the average and the worst computational costs are $O(m\log m)$ and $O(m^2)$, respectively. In Step 2, the initialization of N moth individuals costs time $O(N \times m) = O(m^2)$. In Step 3, the fitness calculation of N moth individuals costs time $O(N)$. In Step 4, Lévy flight operator has time complexity $O(N/2 \times m) = O(m^2)$, straight flight operator has time complexity $O(N/2 \times m) = O(m^2)$, QGROS has time complexity $O(m \times n)$, and sorting the population with Quicksort has average time complexity and worst time complexity of $O(N\log N)$ and $O(N^2)$, respectively. Consequently, the overall computational complexity is $O(m\log m) + O(m^2) + O(N) + O(m^2) + O(m^2) + O(m \times n) + O(N\log N) = O(m^2)$, where m is the number of items and N is the number of moths.

4. Results and Discussion

In this section, we present experimental studies on the proposed discrete MS algorithms for solving SUKP.

Algorithm 2. The main procedure of discrete MS algorithm for SUKP.

Begin
Step 1: Sorting.
Sort all items in S in non-increasing order according to PD_i ($0 \leq i \leq m$), and the indexes of items are recorded in array H [0 . . . m].
Step 2: Initialization.
Set the maximum iteration number *MaxGen* and iteration counter $G = 1$;
$\beta = 1.5$; the acceleration factor $\varphi = 0.618$.
Generate N moth individuals randomly $\{X_1, X_2, \ldots, X_N\}$, $X_i \in [-a, a]^m$.
Divide the whole population into two subpopulations with equal size: subpopulation1 and subpopulation2, according to their fitness.
Calculate the corresponding binary vector $Y_i = T(X_i)$ by using transfer functions ($i = 1, 2, \ldots, N$).
Perform repair and optimization with QGROS.
Step 3: Fitness calculation.
Calculate the initial fitness of each individual, $f(Y_i)$, $1 \leq i \leq N$.
Step 4: While $G <$ *MaxGen* **do**
Update subpopulation 1 by using Lévy flight operator.
Update subpopulation 2 by using fly straightly operator.
Calculate the corresponding binary vector $Y_i = T(X_i)$ by using transfer functions ($i = 1, 2, \ldots, N$).
Perform repair and optimization with QGROS.
Evaluate the fitness of the population and record the $<\mathbf{X_{gbest}}, \mathbf{Y_{gbest}}>$.
$G = G + 1$.
Recombine the two newly-generated subpopulations.
Sort the population by fitness.
Divide the whole population into subpopulation 1 and subpopulation 2.
Step 5: End while
Step 6: Output: the best results.
End.

Test instance: Three groups of thirty SUKP instances were recently presented by He et al. [5]. What needs to be specified is that the set of items $S = \{U_1, U_2, U_3, \ldots, U_m\}$ is represented as a 0–1 matrix M = (r_{ij}), with m rows and n columns. For each element r_{ij} in M ($i = 1, 2, \ldots, m$; $j = 1, 2, \ldots, n$), $r_{ij} = 1$ if and only if $u_j = U_i$. Therefore, each instance contains four factors: (1) m denotes the number of items; (2) n denotes the number of elements; (3) density of element 1 in the matrix M$\alpha \in \{0.1, 0.15\}$; and (4) the ratio of C to the sum of all elements $\beta \in \{0.75, 0.85\}$. According to the relationship between m and n, three types of instances are generated. The first group: 10 SUKP instances with $m > n$, $m \in \{100, 200, 300, 400, 500\}$ and $n \in \{85, 185, 285, 385, 485\}$, named as F01–F10, respectively. The second group: 10 SUKP instances with $m = n$, $m \in \{100, 200, 300, 400, 500\}$ and $n \in \{100, 200, 300, 400, 500\}$, named as S01–S10, respectively. The third group: 10 SUKP instances with $m < n$, $m \in \{85, 185, 285, 385, 485\}$ and $n \in \{100, 200, 300, 400, 500\}$, named as T01–T10, respectively. We selected five instances in each group with $\alpha = 0.1$ and $\beta = 0.75$. The instances can be downloaded at http://sncet.com/ThreekindsofSUKPinstances(EAs).rar. Three categories with different relationships between m and n ($m > n$, $m = n$, and $m < n$) of 15 SUKP instances were selected for testing. The parameters and the best solution value (Best*) [5] are shown in Table 3.

Table 3. The parameters and the best solution value provided in [5] for 15 SUKP instances.

Number	Instance	m	n	Capacity	Best*
1	F01	100	85	12,015	13,251
2	F03	200	185	22,809	13,241
3	F05	300	285	36,126	10,553
4	F07	400	385	50,856	10,766
5	F09	500	485	60,351	11,031
6	S01	100	100	11,223	14,044
7	S03	200	200	25,630	11,846
8	S05	300	300	38,289	12,304
9	S07	400	400	49,822	10,626
10	S09	500	500	63,902	10,755
11	T01	85	100	12,180	11,664
12	T03	185	200	25,405	13,047
13	T05	285	300	38,922	11,158
14	T07	385	400	49,815	10,085
15	T09	485	500	62,516	10,823

Experimental environment: For fair comparisons, all proposed algorithms in this paper were coded in C++ and in the Microsoft Visual Studio 2015 environment. All the experiments were run on a PC with Intel (R) Core (TM) i7-7500 CPU (2.90 GHz and 8.00 GB RAM).

On the stopping condition, we followed the original paper [5] and set the iteration number *MaxGen* equal to $max\{m, n\}$ for all SUKP instances. Here, m denotes the number of items and n is the number of elements in each SUKP instance. In addition, the population size of all the algorithms was set to $N = 20$. For each SUKP instance, we carried out 100 independent replications.

The parameters for the proposed discrete MS algorithms were set as follows: the max step $S_{max} = 1.0$, acceleration factor $\varphi = 0.618$, and the index $\beta = 1.5$.

4.1. The Performance of Discrete MS Algorithm with Different Transfer Functions

Computational results are summarized in Table 4, which records the results for SUKP instances with $m > n$, $m = n$, and $m < n$, respectively. For each instance, we give several criteria to evaluate the comprehensive performance of the twelve discrete MS algorithms. "Best" and "Mean" refer to the best value and the average value for each instance obtained by each algorithm among 100 independent runs. The best solution provided in [5] are given in parentheses in the first column.

Table 4. The best values and average values of twelve discrete MS algorithms on 15 SUKP instances.

Number	Criterion	MSS1	MSS2	MSS3	MSS4	MSV1	MSV2	MSV3	MSV4	MSO1	MSO2	MSO3	MSO4
F01	Best	12,698	13,283	12,861	13,057	13,003	13,003	13,044	13,044	12,973	12,678	13,283	13,283
(13251)	Mean	12,250	13,102	12,168	12,227	12,564	12,740	12,858	12,697	12,320	12,066	13,052	13,062
F03	Best	12,762	13,286	12,216	12,189	12,875	12,639	11,953	12,267	13,175	12,255	13,322	13,521
(13241)	Mean	11,777	12,860	11,465	11,261	12,176	12,100	11,321	11,193	12,371	11,007	13,101	13,193
F05	Best	10,142	10,668	9974	10,047	9966	9562	9752	9460	10,539	9656	10,643	11,127
(10553)	Mean	9588	10,196	9465	9393	9352	9120	9250	9131	9822	8987	10,381	10,302
F07	Best	10,456	11,321	9793	10,005	10,625	9539	9917	9814	10,906	9801	11,321	11,435
(10766)	Mean	9750	10,644	9349	9467	10,042	9150	9317	9265	10,141	9028	10,833	10,411
F09	Best	10,669	11,410	10,642	10,461	10,718	10,725	10,598	10,288	11,279	9808	11,172	11,031
(11031)	Mean	10,293	10,913	10,082	9965	10,420	9969	10,134	9997	10,648	9429	10,750	10,716
S01	Best	13,405	14,044	13,080	13,611	13,396	13,814	13,721	13,721	13,659	13,202	14,003	14,044
(14044)	Mean	12,725	13,478	12,418	12,607	13,211	13,569	13,540	13,503	12,899	12,339	13,583	13,649
S03	Best	11,249	11,104	10,904	11,295	11,329	10,802	10,481	10,808	11,757	11,147	11,873	12,350
(11846)	Mean	10,469	10,576	10,282	10,285	10,622	9879	10,112	10,212	10,789	9975	11,419	11,508
S05	Best	11,649	12,071	11,472	11,459	11,799	11,686	11,421	11,380	11,862	11,048	12,240	12,598
(12304)	Mean	10,979	11,650	10,753	10,787	11,199	11,206	10,898	11,165	11,272	10,153	11,721	11,541
S07	Best	10,330	10,990	10,218	10,073	10,177	9669	9957	9977	10,650	10,006	10,722	10,727
(10626)	Mean	9831	10,379	9766	9681	9968	9286	9372	9460	10,019	9426	10,327	10,343
S09	Best	10,074	10,495	9995	10,037	9938	10,025	10,043	10,052	10,199	9553	10,355	10,355
(10755)	Mean	9719	9968	9583	9675	9654	9707	9804	9713	9807	9,127	10,056	9919
T01	Best	11,034	11,573	11,158	11,332	11,427	11,027	11,151	11,076	11,195	11,159	11,519	11,735
(11664)	Mean	10,577	11,259	10,491	10,501	10,812	10,572	10,496	10,781	10,568	10,495	11,276	11,287
T03	Best	12,234	13,306	12,357	12,136	12,415	12,633	12,039	11,829	12,798	12,085	13,378	13,647
(13047)	Mean	11,549	12,621	11,624	11,522	11,745	11,743	11,535	11,400	11,980	11,267	12,948	13,000
T05	Best	11,025	11,173	11,167	10,765	10,814	10,725	10,485	11,240	11,183	10,299	11,226	11,391
(11158)	Mean	10,385	10,871	10,247	10,118	10,635	10,229	10,059	10,735	10,651	9584	10,957	10,816
T07	Best	9676	9609	9140	9169	9594	9303	9049	8965	9675	9198	9783	9739
(10085)	Mean	8987	9264	8873	8875	9161	9131	8642	8733	9154	8694	9261	9240
T09	Best	10,208	10,549	10,131	10,094	10,115	10,201	9866	10,005	10,450	9989	10,660	10,539
(10823)	Mean	9856	10,205	9753	9711	9949	9771	9506	9714	9985	9332	10,350	10,190

In Table 4, it can be easily observed that MSO4 outperforms the eleven other discrete MS algorithms and demonstrates the best comprehensive performance when solving all fifteen SUKP instances. In addition, MSS2 and MSO3 show comparable performance.

To evaluate the performance of each algorithm, the relative percentage deviation (*RPD*) was defined to represent the similarity between the best value obtained by each algorithm and the best solution 5. The *RPD* of each SUKP instance is calculated as follows.

$$RPD = (\text{Best}^* - \text{Best})/\text{Best}^* \times 100 \tag{16}$$

where Best^* is the best solution provided in [5]. Clearly, if the value of *RPD* is less than 0, the algorithm updates the best solution of the SUKP test instance in [5]. The statistical results are shown in Table 5.

Table 5. The effect of twelve transfer functions on the performance of discrete MS algorithm (*RPD* values).

Number	MSS1	MSS2	MSS3	MSS4	MSV1	MSV2	MSV3	MSV4	MSO1	MSO2	MSO3	MSO4
F01	4.17	−0.24	2.94	1.46	1.87	1.87	1.56	1.56	2.10	4.32	−0.24	−0.24
F03	3.62	−0.34	7.74	7.95	2.76	4.55	9.73	7.36	0.50	7.45	−0.61	−2.11
F05	3.89	−1.09	5.49	4.79	5.56	9.39	7.59	10.36	0.13	8.50	−0.85	−5.44
F07	2.88	−5.16	9.04	7.07	1.31	11.40	7.89	8.84	−1.30	8.96	−5.16	−6.21
F09	3.28	−3.44	3.53	5.17	2.84	2.77	3.93	6.74	−2.25	11.09	−1.28	0.00
S01	4.55	0.00	6.86	3.08	4.61	1.64	2.30	2.30	2.74	6.00	0.29	0.00
S03	5.04	6.26	7.95	4.65	4.36	8.81	11.52	8.76	0.75	5.90	−0.23	−4.25
S05	5.32	1.89	6.76	6.87	4.10	5.02	7.18	7.51	3.59	10.21	0.52	−2.39
S07	2.79	−3.43	3.84	5.20	4.23	9.01	6.30	6.11	−0.23	5.83	−0.90	−0.95
S09	6.33	2.42	7.07	6.68	7.60	6.79	6.62	6.54	5.17	11.18	3.72	3.72
T01	5.40	0.78	4.34	2.85	2.03	5.46	4.40	5.04	4.02	4.33	1.24	−0.61
T03	6.23	−1.99	5.29	6.98	4.84	3.17	7.73	9.34	1.91	7.37	−2.54	−4.60
T05	1.19	−0.13	−0.08	3.52	3.08	3.88	6.03	−0.73	−0.22	7.70	−0.61	−2.09
T07	4.06	4.72	9.37	9.08	4.87	7.75	10.27	11.11	4.07	8.80	2.99	3.43
T09	5.68	2.53	6.39	6.74	6.54	5.75	8.84	7.56	3.45	7.71	1.51	2.62
Mean	4.30	0.19	5.77	5.47	4.04	5.82	6.79	6.56	1.63	7.69	−0.14	−1.28

In Table 5, it can be seen that, in all twelve discrete MS algorithms, MSS2, MSS3, MSV4, MSO1, MSO3, and MSO4 all update the best solutions [5]. However, MSS3 and MSV4 update only one SUKP instance, T05. MSO1 updates the instances F07, F09, S07, and T05. Moreover, MSO4 still keeps the best performance because its total average *RPD* is only −1.28. The total average *RPD* of MSO3 is −0.14, which implies that MSO3 is slightly worse than MSO4 but outperforms the ten other discrete MS algorithms. Obviously, MSS2 is the third best of the twelve discrete MS algorithms. Indeed, it can also be seen that MSO4 updates and obtains the best solutions [5] ten and two times (out of 15), i.e., 66.67% and 13.33% of the whole instance set, respectively. MSO3 updates and fails to find the best solutions 5 nine and six times (out of 15), i.e., 60.00% and 40.00% of the whole instance set, respectively. MSS2 updates and obtains the best solutions 5 eight (53.33%) and one times (6.60%), respectively.

To further evaluate the comprehensive performance of twelve discrete MS algorithms in solving fifteen SUKP instances, the average ranking based on the best values are displayed in Table 6 and Figure 3, respectively. In Table 6 and Figure 3, the average ranking value of MSO4 is 1.60 and it still ranks first. In addition, MSO3 and MSS2 are the second and the third best algorithms, respectively, which is very consistent with the previous analysis. The ranking of twelve discrete MS algorithms based on the best values are as follows:

$$\begin{aligned} &\text{MSO4} \succ \text{MSO3} \succ \text{MSS2} \succ \text{MSO1} \succ \text{MSS1} \succ \text{MSV1} \succ \text{MSS4} \succ \text{MSS3} \\ &= \text{MSV2} \succ \text{MSV4} \succ \text{MSV3} \succ \text{MSO2} \end{aligned} \tag{17}$$

Table 6. Ranks of twelve discrete MS algorithms based on the best values.

Number	MSS1	MSS2	MSS3	MSS4	MSV1	MSV2	MSV3	MSV4	MSO1	MSO2	MSO3	MSO4
F01	11	1	10	4	7	7	5	5	9	12	1	1
F03	6	3	10	11	5	7	12	8	4	9	2	1
F05	5	2	7	6	8	11	9	12	4	10	3	1
F07	5	1	6	8	7	12	11	10	4	9	3	2
F09	7	1	8	10	6	5	9	11	2	12	3	4
S01	9	1	12	8	10	4	5	5	7	11	3	1
S03	6	8	9	5	4	11	12	10	3	7	2	1
S05	7	3	8	9	5	6	10	11	4	12	2	1
S07	5	1	6	8	7	12	11	10	4	9	3	2
S09	6	1	10	8	11	9	7	6	4	12	2	2
T01	11	2	8	5	4	12	9	10	6	7	3	1
T03	8	3	7	9	6	5	11	12	4	10	2	1
T05	7	5	6	9	8	10	11	2	4	12	3	1
T07	3	5	10	9	6	7	11	12	4	8	1	2
T09	5	2	7	9	8	6	12	10	4	11	1	3
Mean	6.67	2.60	8.27	7.87	6.80	8.27	9.67	8.93	4.47	10.07	2.27	1.60

By looking closely at Figures 2 and 3, it is not difficult to see that V3, V4, and O2 exhibit the worst performance, which is consistent in the two figures. Similar to the previous analysis in Figure 2, O1, O3, and O4 show satisfactory performance among 12 transfer functions. Thus, it can be inferred that *PR* value can be used as a criterion for selecting transfer functions.

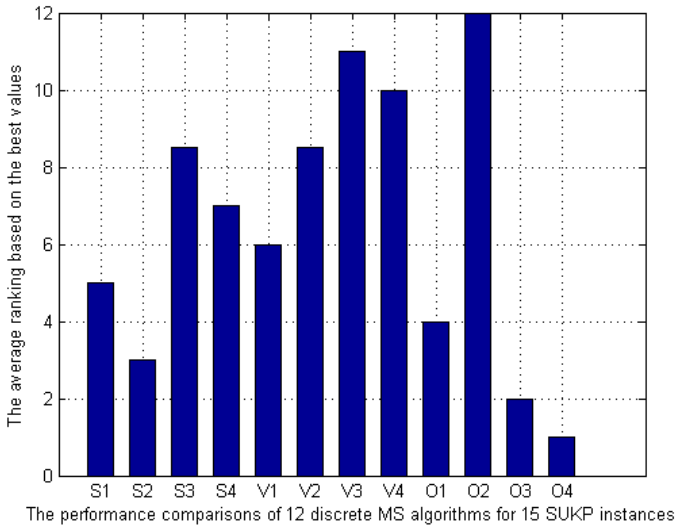

Figure 3. Comparison of the average rank of 12 discrete MS algorithms for 15 SUKP instances.

To analyze the experimental results for statistical purposes, we selected three representative instances (F09, S09, and T09) and provided boxplots in Figures 4–6. In Figure 4, the boxplot of MSS2 has greater value and less height than those of other eleven algorithms. In Figures 5 and 6, MSO3 exhibits a similar phenomenon as MSS2 in Figure 4. Additionally, the performance of MSO2 is the worst. In Figures 4–6, we can also observe that MSO3 performs slightly better than MSO4 in solving large-scale instances.

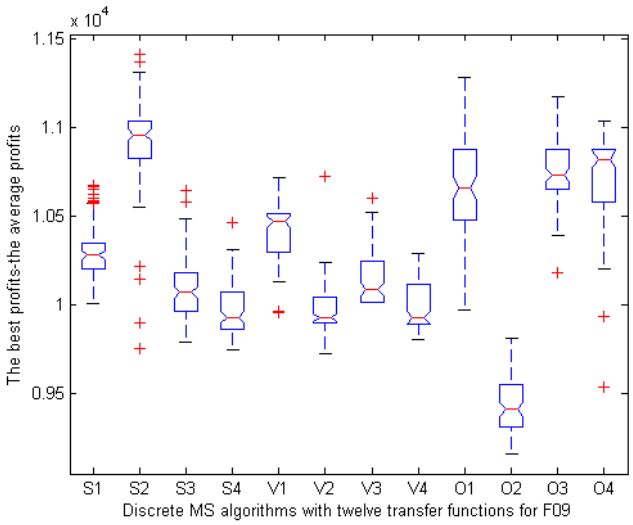

Figure 4. Boxplot of the best values on F09 in 100 runs.

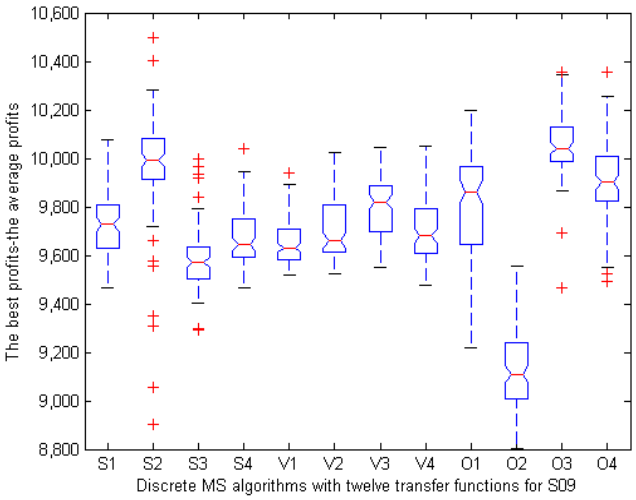

Figure 5. Boxplot of the best values on S09 in 100 runs.

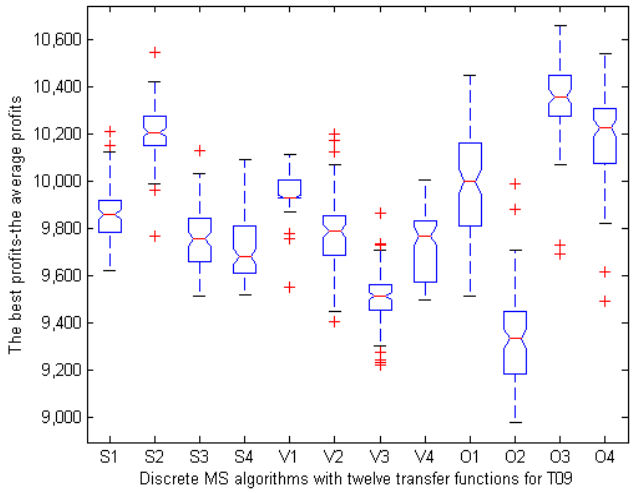

Figure 6. Boxplot of the best values on T09 in 100 runs.

Moreover, optimization process of each algorithm in solving F09, S09, and T09 instances is given in Figures 7–9, respectively. In these three figures, all the function values are the average best values achieved from 100 runs. In Figure 7, the initial value of MSS2 is greater than that of other algorithms

and then it quickly converges to the global optimum. For MSO3, the same scene appears in Figures 8 and 9. Overall, MSS2 and MSO3 have stronger optimization ability and faster convergence speed than the other discrete MS algorithms.

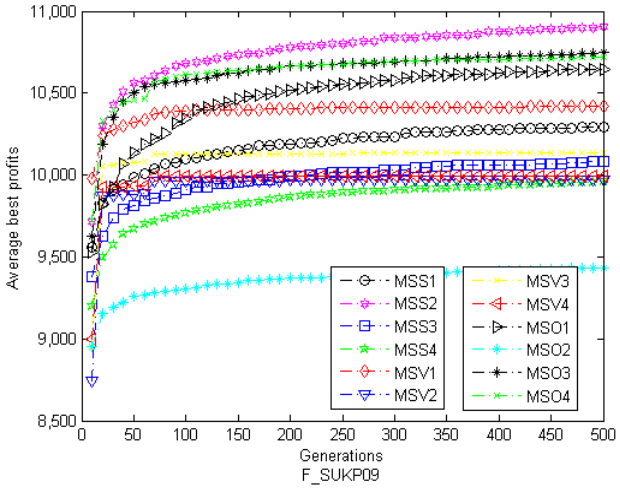

Figure 7. The convergence graph of twelve discrete MS algorithms on F09.

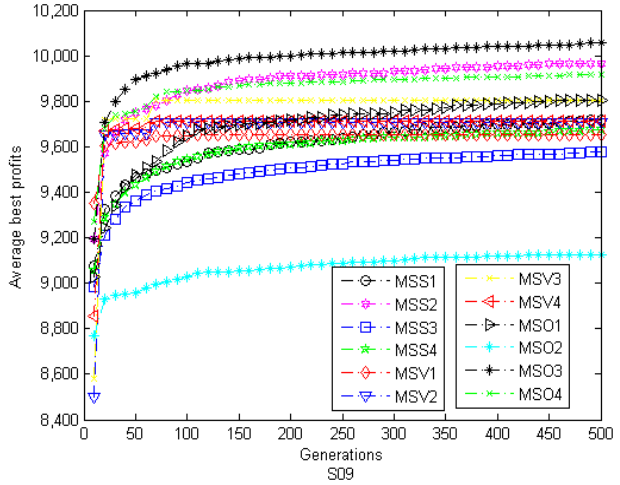

Figure 8. The convergence graph of twelve discrete MS algorithms on S09.

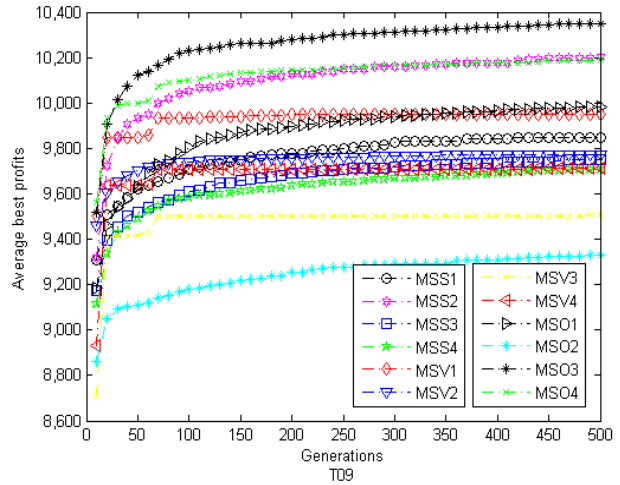

Figure 9. The convergence graph of twelve discrete MS algorithms on T09.

Through the above experimental analysis, the following conclusions can be drawn: (1) For S-shaped transfer functions, the combination of S2 and MS (MSS2) is the most effective. (2) As far as V-shaped transfer functions are concerned, the combination of V1 and MS (MSV1) shows the best performance. (3) In the case of other shapes transfer functions, the more effective algorithms are MSO4, MSO3, and MSO1. (4) By comparing the family of S-shaped transfer functions and V-shaped transfer functions, the family of S-shaped transfer functions with MS is suitable for solving SUKP problem. (5) MSO4 has advantages over other algorithms in terms of the quality of solutions. (6) As far as the stability and convergence rate are concerned, MSO3 and MSS2 perform better than other algorithms.

Overall, it is evident that MSO4 has the best results (considering *RPD* values and average ranking values) on fifteen SUKP instances. Therefore, it appears that the proposed other-shapes family of transfer functions, particularly the O4 function, has many advantages combined with other algorithms to solve binary optimization problems. Additionally, the O3 function and S2 function are also suitable functions that can be considered for selection. In brief, these results demonstrate that the transfer function plays a very important role in solving SUKP using discrete MS algorithm. Thus, by carefully selecting the appropriate transfer function, the performance of discrete MS algorithm can be improved obviously.

4.2. Estimation of the Solution Space

SUKP is a binary coded problem and the solution space can be represented as a graph $G = (V, E)$, in which vertex set $V = S$, where S is the set of solutions for a SUKP instance, $S = \{0, 1\}^n$ and edge set $E = \{(s, s') \in S \times S | d(s, s') = d_{min}\}$, where d_{min} is the minimum distance between two points in the search space. Especially, hamming distance is used to describe the similarity between individuals. Obviously, the minimum distance is 0 when all bits have the same value and the maximum distance is n, where n is the dimension of SUKP instance.

Here, MSO4 is specially selected to analyze the solution space for F01, S01, and T01 SUKP instance. The distribution of fitness at generation 0 and generation 100 is presented in Figures 10–12. The distance between each individual and the best individual is given in Figures 13–15. In Figures 10–12, we can see that, at generation 0, the fitness values are more dispersed and worse than that at generation 100. In Figure 13, it can be observed that the hamming distance varies from 0 to 35 at generation 0 while the range is 0 to 12 at generation 100. Moreover, the hamming distance can be divided into eight levels at generation 100, which demonstrates that all individuals tend to some superior individuals. However, this phenomenon is not evident in S01 and T01.

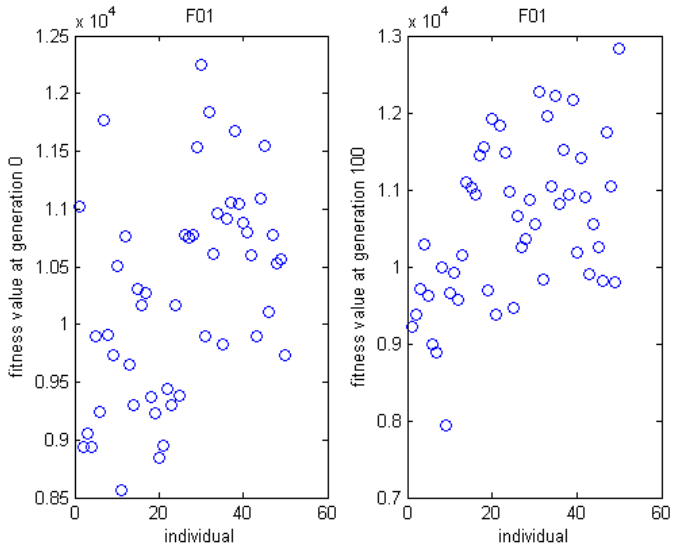

Figure 10. The distribution graph of fitness on MSO4 for F01.

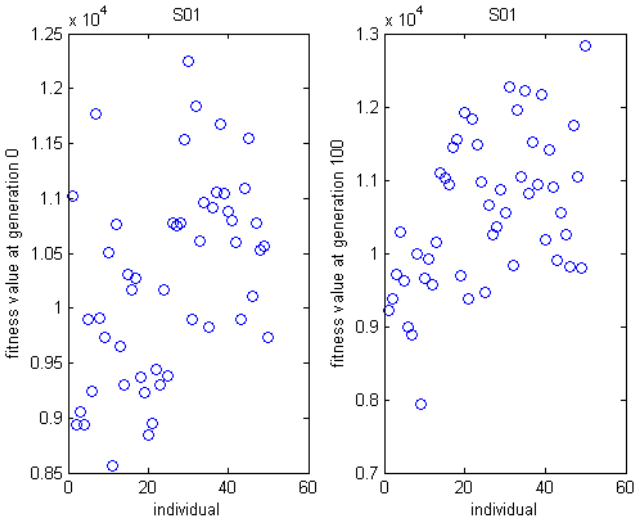

Figure 11. The distribution graph of fitness on MSO4 for S01.

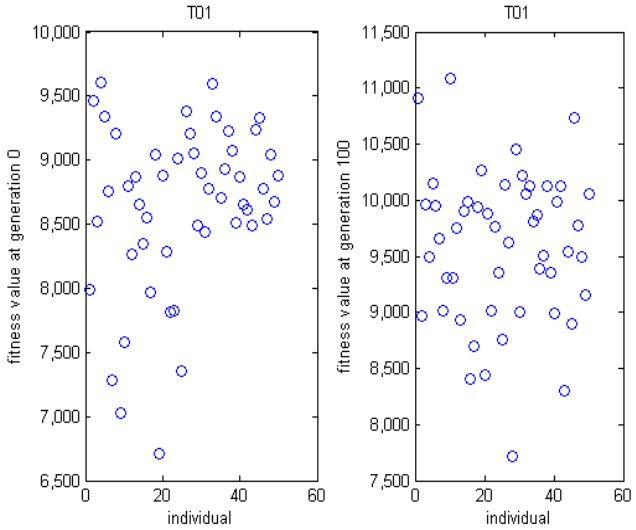

Figure 12. The distribution graph of fitness on MSO4 for T01.

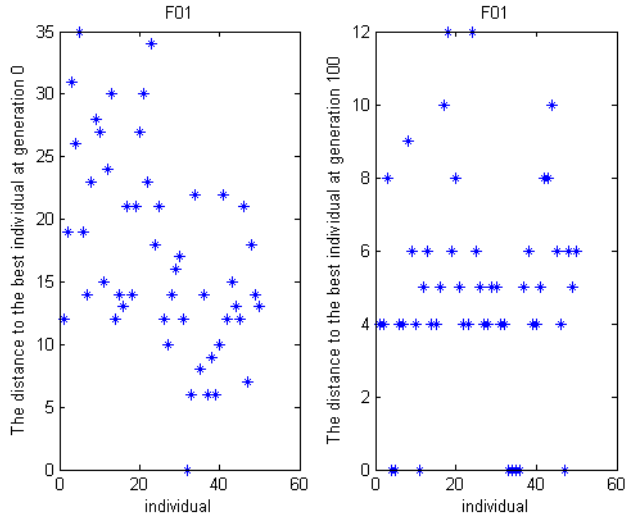

Figure 13. The distance to the best individual for F01.

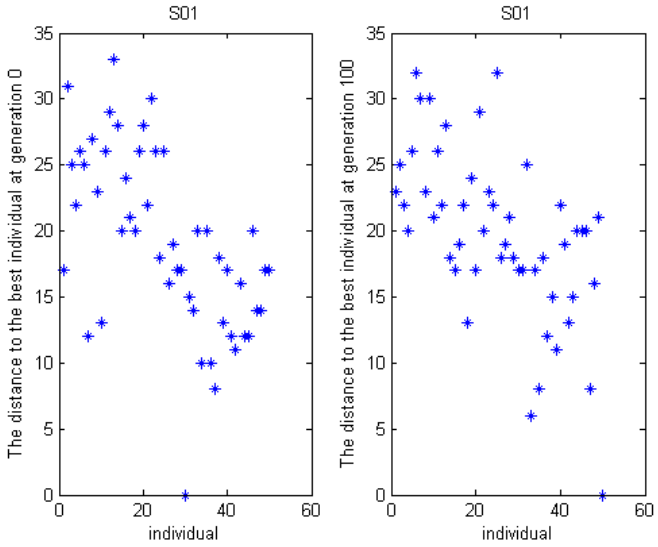

Figure 14. The distance to the best individual for S01.

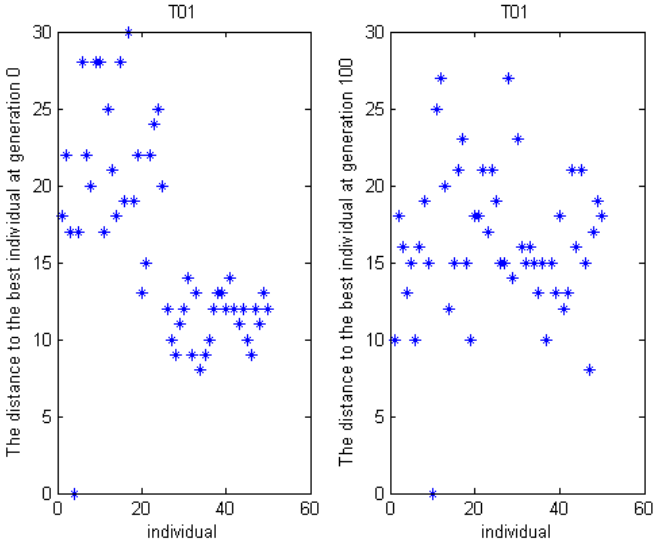

Figure 15. The distance to the best individual for T01.

To intuitively understand the similarity of the solutions, the spatial structure of the solutions at generation 100 is illustrated in Figures 16–18. In Figure 16, the first node (denoting the first individual) has the maximum degree which also shows more individuals have approached the better individual. However, the value of degree is not much different in Figures 17 and 18. This result is consistent with the previous analysis.

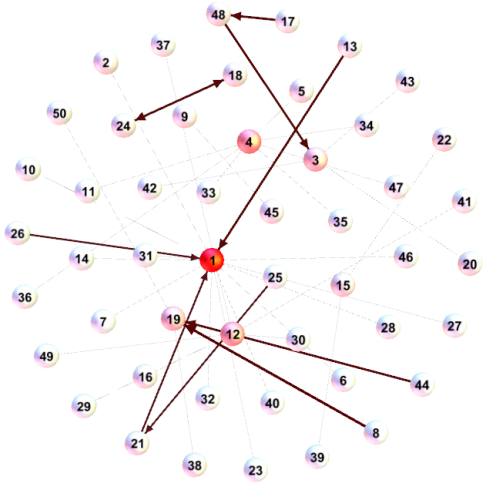

Figure 16. The spatial structure graph for F01 at generation 100.

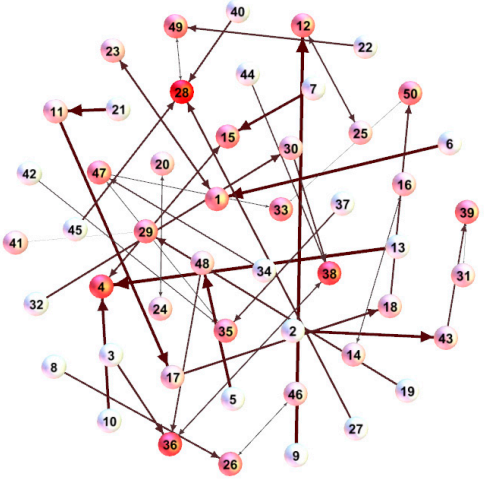

Figure 17. The spatial structure graph for S01 at generation 100.

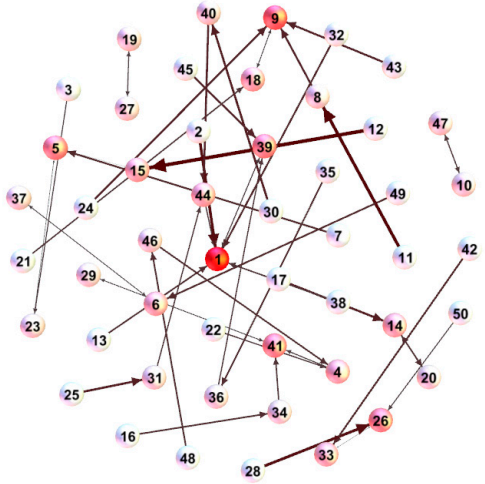

Figure 18. The spatial structure graph for T01 at generation 100.

4.3. Discrete MS Algorithm vs. Other Optimization Algorithms

To further verify the performance of discrete MS algorithm, we chose MSO4 algorithm to compare with five other optimization algorithms. These comparison algorithms include PSO [12], DE [8],

global harmony search (GHS) [30], firefly algorithm (FA) [31], and monarch butterfly optimization (MBO) [32,33]. In DE, the DE/rand/1/bin scheme was adopted. PSO, FA, and MBO are classical or novel swarm intelligence algorithms that simulate the social behavior of birds, firefly, and monarch butterfly, respectively. DE is derived from evolutionary theory in nature and has been proved to be one of the most promising stochastic real-value optimization algorithms. GHS is an efficient variant of HS, which imitates the music improvisation process. It is also noteworthy that all five comparison algorithms adopt the discretization method introduced in this paper and combine with O4, respectively. The parameter setting for each algorithm are shown in Table 7.

Table 7. The parameter settings of six algorithms on SUKP.

Algorithm	Parameters	Value
PSO	*Cognitive constant C1*	1.0
	Social constant C2	1.0
	Inertial constant *W*	0.3
DE	*Weighting factor F*	0.9
	Crossover constant CR	0.3
GHS	*Harmony memory considering rate HMCR*	0.9
	Pitch adjusting rate PAR	0.3
FA	*Alpha*	0.2
	Beta	1.0
	Gamma	1.0
MBO	*Migration ratio*	3/12
	Migration period	1.4
	Butterfly adjusting rate	1/12
	Max step	1.0
MSO4	*Max step* S_{max}	1.0
	Acceleration factor φ	0.618
	Lévy distribution parameter β	1.5

The best results and average results obtained by six methods over 100 independent runs as well the average time cost of each computation (unit: second, represented as "time") are summarized in Table 8. The frequency (T_{Best} and T_{Mean}) and average ranking (R_{Best} and R_{Mean}) of each algorithm with the best performance based on the best values and average values are also recorded in Table 8. The average time cost of each computation for solving fifteen SUKP instances is illustrated in Figure 19. In Table 8, on best, MSO4 outperforms other methods on eight of fifteen instances (F01, F03, F05, F07, S01, S03, S05, and T03). MBO is the second most effective. In terms of average ranking, there is little difference between the performance of MSO4 and MBO. In terms of the average time cost, it can be observed in Figure 19 that DE has the slowest computing speed. However, GHS has surprisingly fast solving speed. In addition, MSO4 is second among the six algorithms. Overall, the computing speed of PSO, FA, MBO and MSO4 shows little difference.

Table 8. Computational results and comparisons on the Best and Mean on 15 SUKP instances.

Number	Criterion	PSO	DE	GHS	FA	MBO	MSO4
F01	Best	13,283	13,125	13,251	13,283	13,283	13,283
(13251)	Mean	12,981	12,923	12,492	13,041	12,941	13,062
	Time	1.297	1.923	0.330	1.627	3.684	1.398
F03	Best	13,319	13,172	12,323	13,282	13,381	13,521
(13241)	Mean	12,697	12,443	11,231	12,544	12,886	13,193
	Time	8.643	13.381	0.603	11.552	11.676	7.901
F05	Best	10,408	10,214	10,512	10,191	10,786	11,127
(10553)	Mean	9825	9420	10,179	9092	10,210	10,302
	Time	28.628	41.633	0.928	45.647	40.938	24.912
F07	Best	11,091	10,135	11,255	9740	11,142	11,435
(10766)	Mean	10,613	9573	10,642	9226	10,463	10,411
	Time	63.290	124.504	1.637	97.102	61.588	56.838
F09	Best	11,046	11,016	11,536	11,099	11,546	11,031
(11031)	Mean	10,473	10,443	11,199	10,473	10,736	10,716
	Time	138.551	225.749	3.179	170.035	157.478	124.378
S01	Best	13,814	13,519	13,522	13,814	14,044	14,044
(14044)	Mean	13,575	12,964	12,656	13,472	13,612	13,649
	Time	1.608	2.800	0.358	1.805	2.617	1.646
S03	Best	11,914	11,085	11,531	11,406	11,955	12,350
(11846)	Mean	10,978	10,408	10,925	10,833	11,056	11,508
	Time	8.437	14.543	0.476	10.753	9.371	8.112
S05	Best	12,574	12,071	12,104	11,398	12,369	12,598
(12304)	Mean	11,709	11,251	11,492	10,993	11,604	11,541
	Time	35.259	46.302	1.014	37.130	26.551	28.612
S07	Best	10,669	10,267	10,952	10,241	10,906	10,727
(10626)	Mean	10,217	9753	10,497	9827	10,237	10,343
	Time	79.622	101.118	1.718	77.458	76.049	58.433
S09	Best	10,352	10,100	10,434	10,057	10,633	10,355
(10755)	Mean	10,104	9708	10,239	9766	10,139	9919
	Time	144.377	242.428	3.013	167.492	153.835	121.622
T01	Best	11,752	11,469	11,434	11,755	11,748	11,735
(11664)	Mean	11,152	10,930	10,370	11,226	11,207	11,287
	Time	1.517	1.892	0.407	1.722	1.668	1.354
T03	Best	13,100	9624	12,618	11,487	13,008	13,647
(13047)	Mean	12,091	9,122	11,855	10,880	12,189	13,000
	Time	7.964	12.708	0.507	11.958	10.702	7.642
T05	Best	11,032	10,669	11,071	11,557	11,090	11,391
(11158)	Mean	10,656	10,490	10,722	10,983	10,686	10,816
	Time	31.753	41.176	0.822	32.175	25.044	24.539
T07	Best	9790	9250	9857	9,392	9770	9739
(10085)	Mean	9636	8897	9447	8,895	9322	9240
	Time	62.079	113.779	1.639	79.984	67.675	57.000
T09	Best	10,482	10,260	10,643	10,207	10,661	10,539
(10823)	Mean	10,111	9717	10,306	9783	10,249	10,190
	Time	121.926	195.754	2.896	144.926	136.766	114.066
T_{Best}		1	0	2	3	5	8
T_{Mean}		2	0	5	1	0	6
R_{Best}		3.27	5.47	3.40	4.40	2.20	2.27
R_{Mean}		3.17	5.47	3.27	4.43	2.53	2.13

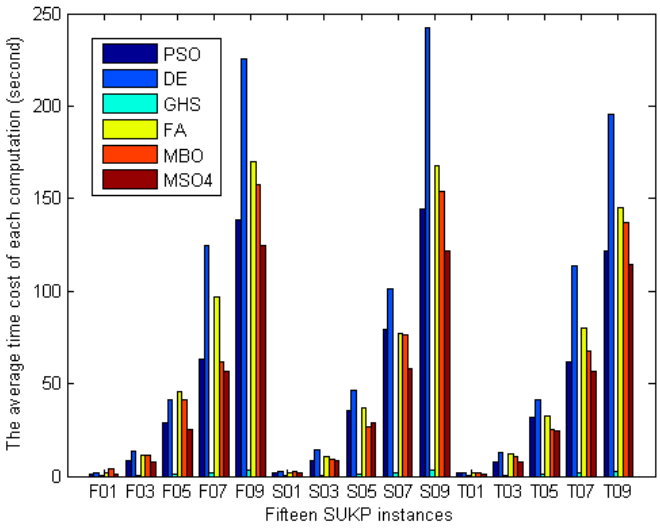

Figure 19. The average time cost of each computation for solving fifteen SUKP instances.

To investigate the difference between the results obtained by MSO4 and those by the comparison algorithm from the perspective of statistics, Wilcoxon's rank sum tests with the 5% significance level were performed. The results of rank sum tests are recorded in Table 9. In Table 9, "1" and "−1" indicate that MSO4 is superior or inferior to the corresponding comparison algorithm, respectively, while "0" shows that there is no statistical difference at 5% significance level between the two comparison algorithms. The statistical result is shown in Table 9.

In Table 8, MSO4 outperforms PSO and DE on all fifteen instances. In addition, MSO4 performs better than GHS and FA on most of the instances except for S05 and F01, respectively. Meanwhile, MSO4 is superior to MBO on eleven instances except for F07, F09, S01, and S05. Statistically, there is no difference between the performance of MSO4 and that of MBO for these four instances.

Considering the results shown in Tables 8 and 9, a conclusion can be drawn that the performance of MSO4 is superior to or at least quite competitive with the five other methods.

Table 9. Results of rank sum tests for MSO4 with the comparison algorithms.

MSO4	PSO	DE	GHS	FA	MBO
F01	1	1	1	0	1
F03	1	1	1	1	1
F05	1	1	1	1	1
F07	1	1	1	1	0
F09	1	1	1	1	0
S01	1	1	1	1	0
S03	1	1	1	1	1
S05	1	1	0	1	0
S07	1	1	1	1	1
S09	1	1	1	1	1
T01	1	1	1	1	1
T03	1	1	1	1	1
T05	1	1	1	1	1
T07	1	1	1	1	1
T09	1	1	1	1	1
1	15	15	14	14	11
0	0	0	1	1	4
−1	0	0	0	0	0

5. Conclusions

In this paper, twelve different transfer functions-based discrete MS algorithms are proposed for solving SUKP. These transfer functions can be divided into three families, S-shaped, V-shaped, and other-shaped transfer functions. To investigate the performance of twelve discrete MS algorithms, three groups of fifteen SUKP instances were employed and the experimental results were compared and analyzed comprehensively. From the experimental results, we found that MSO4 has the best performance. Furthermore, the relative percentage deviation (*RPD*) was calculated to evaluate the similarity between the best value obtained by each algorithm and the best solution provided in [5]. The results show that six algorithms update the best solutions [5] for 11 SUKP instances. The results also indicate that four other shapes transfer functions, especially the O4 function combined with MS, have merits for solving discrete optimization problems.

The comparison results on the fifteen SUKP instances among MSO4 and five state-of-the-art algorithms show that MSO4 performs competitively.

There are several possible directions for further study. First, we will investigate some new transfer functions on other algorithms such as krill herd algorithm (KH) [34–38], fruit fly optimization algorithm (FOA) [39], earthworm optimization algorithm (EWA) [40], and cuckoo search (CS) [41,42]. Second, we will study other techniques to discrete continuous optimization algorithms such as k-means framework [43]. Third, we will apply these twelve transfer functions-based discrete MS algorithms to other related and more complicated binary optimization problems including multidimensional knapsack problem (MKP) [39] and flow shop scheduling problem (FSSP) [44]. Finally, we will incorporate other strategies, namely, information feedback [45] and chaos theory [46], into MS to improve the performance of the algorithm.

Author Contributions: Writing and methodology, Y.F.; supervision, H.A.; review and editing, X.G.

Funding: This research was funded by National Natural Science Foundation of China, grant number 61806069, Key Research and Development Projects of Hebei Province, grant number 17210905.

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

References

1. Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. *Introduction to Algorithms*; MIT Press: Cambridge, MA, USA, 2009.
2. Du, D.Z.; Ko, K.I. *Theory of Computational Complexity*; John Wiley & Sons: Hoboken, NJ, USA, 2011.
3. Goldschmidt, O.; Nehme, D.; Yu, G. Note: On the set-union knapsack problem. *Naval Res. Logist. (NRL)* **1994**, *41*, 833–842. [CrossRef]
4. Arulselvan, A. A note on the set union knapsack problem. *Discret. Appl. Math.* **2014**, *169*, 214–218. [CrossRef]
5. He, Y.; Xie, H.; Wong, T.L.; Wang, X. A novel binary artificial bee colony algorithm for the set-union knapsack problem. *Future Gener. Comput. Syst.* **2017**, *78*, 77–86. [CrossRef]
6. Yang, X.; Vernitski, A.; Carrea, L. An approximate dynamic programming approach for improving accuracy of lossy data compression by Bloom filters. *Eur. J. Oper. Res.* **2016**, *252*, 985–994. [CrossRef]
7. Schneier, B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*; John Wiley & Sons: Hoboken, NJ, USA, 2007.
8. Engelbrecht, A.P.; Pampara, G. Binary differential evolution strategies. In Proceedings of the IEEE Congress on Evolutionary Computation, Singapore, 25–28 September 2007; pp. 1942–1947.
9. Ozsoydan, F.B.; Baykasoglu, A. A swarm intelligence-based algorithm for the set-union knapsack problem. *Future Gener. Comput. Syst.* **2018**, *93*, 560–569. [CrossRef]
10. Wang, G.G. Moth search algorithm: A bio-inspired metaheuristic algorithm for global optimization problems. *Memetic Comput.* **2016**. [CrossRef]
11. Feng, Y.; Wang, G.G. Binary moth search algorithm for discounted 0-1 knapsack problem. *IEEE Access* **2018**, *6*, 10708–10719. [CrossRef]

12. 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—Computational Cybernetics and Simulation, Orlando, FL, USA, 12–15 October 1997; Volume 5, pp. 4104–4108.
13. Karthikeyan, S.; Asokan, P.; Nickolas, S.; Page, T. A hybrid discrete firefly algorithm for solving multi-objective flexible job shop scheduling problems. *Int. J. Bio-Inspired Comput.* **2015**, *7*, 386–401. [CrossRef]
14. 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]
15. Rashedi, E.; Nezamabadi-Pour, H.; Saryazdi, S. BGSA: Binary gravitational search algorithm. *Nat. Comput.* **2010**, *9*, 727–745. [CrossRef]
16. Saremi, S.; Mirjalili, S.; Lewis, A. How important is a transfer function in discrete heuristic algorithms. *Neural Comput. Appl.* **2015**, *26*, 625–640. [CrossRef]
17. Mirjalili, S.; Lewis, A. S-shaped versus V-shaped transfer functions for binary particle swarm optimization. *Swarm Evol. Comput.* **2013**, *9*, 1–14. [CrossRef]
18. Pampara, G.; Franken, N.; Engelbrecht, A.P. Combining particle swarm optimisation with angle modulation to solve binary problems. In Proceedings of the 2005 IEEE Congress on Evolutionary Computation, Edinburgh, UK, 2–5 September 2005; Volume 1, pp. 89–96.
19. Leonard, B.J.; Engelbrecht, A.P.; Cleghorn, C.W. Critical considerations on angle modulated particle swarm optimisers. *Swarm Intell.* **2015**, *9*, 291–314. [CrossRef]
20. Costa, M.F.P.; Rocha, A.M.A.C.; Francisco, R.B.; Fernandes, E.M.G.P. Heuristic-based firefly algorithm for bound constrained nonlinear binary optimization. *Adv. Oper. Res.* **2014**, *2014*, 215182. [CrossRef]
21. Burnwal, S.; Deb, S. Scheduling optimization of flexible manufacturing system using cuckoo search-based approach. *Int. J. Adv. Manuf. Technol.* **2013**, *64*, 951–959. [CrossRef]
22. Pampará, G.; Engelbrecht, A.P. Binary artificial bee colony optimization. In Proceedings of the 2011 IEEE Symposium on Swarm Intelligence (SIS), Paris, France, 11–15 April 2011; pp. 1–8.
23. Zhu, H.; He, Y.; Wang, X.; Tsang, E.C.C. Discrete differential evolutions for the discounted {0-1} knapsack problem. *Int. J. Bio-Inspired Comput.* **2017**, *10*, 219–238. [CrossRef]
24. Changdar, C.; Mahapatra, G.S.; Pal, R.K. An improved genetic algorithm based approach to solve constrained knapsack problem in fuzzy environment. *Expert Syst. Appl.* **2015**, *42*, 2276–2286. [CrossRef]
25. Lim, T.Y.; Al-Betar, M.A.; Khader, A.T. Taming the 0/1 knapsack problem with monogamous pairs genetic algorithm. *Expert Syst. Appl.* **2016**, *54*, 241–250. [CrossRef]
26. Cao, L.; Xu, L.; Goodman, E.D. A neighbor-based learning particle swarm optimizer with short-term and long-term memory for dynamic optimization problems. *Inf. Sci.* **2018**, *453*, 463–485. [CrossRef]
27. Chih, M. Three pseudo-utility ratio-inspired particle swarm optimization with local search for multidimensional knapsack problem. *Swarm Evol. Comput.* **2017**, *39*. [CrossRef]
28. Michalewicz, Z.; Nazhiyath, G. Genocop III: A co-evolutionary algorithm for numerical optimization problems with nonlinear constraints. In Proceedings of the 1995 IEEE International Conference on Evolutionary Computation, Perth, Western Austrilia, 29 November–1 December 1995; Volume 2, pp. 647–651.
29. Liu, X.J.; He, Y.C.; Wu, C.C. Quadratic greedy mutated crow search algorithm for solving set-union knapsack problem. Microelectro. Comput. 2018, 35, 13–19.
30. Omran, M.G.H.; Mahdavi, M. Global-best harmony search. *Appl. Math. Comput.* **2008**, *198*, 643–656. [CrossRef]
31. Yang, X.S. Firefly Algorithm, Lévy Flights and Global Optimization. In *Research and Development in Intelligent Systems XXVI*; Springer: London, UK, 2010; pp. 209–218.
32. Wang, G.-G.; Deb, S.; Cui, Z. Monarch butterfly optimization. *Neural Comput. Appl.* **2015**, 1–20. [CrossRef]
33. Feng, Y.; Wang, G.G.; Li, W.; Li, N. Multi-strategy monarch butterfly optimization algorithm for discounted {0-1} knapsack problem. *Neural Comput. Appl.* **2017**, 1–18. [CrossRef]
34. Wang, G.-G.; Gandomi, A.H.; Alavi, A.H. An effective krill herd algorithm with migration operator in biogeography-based optimization. *Appl. Math. Model.* **2014**, *38*, 2454–2462. [CrossRef]
35. Wang, G.-G.; Gandomi, A.H.; Alavi, A.H. Stud krill herd algorithm. *Neurocomputing* **2014**, *128*, 363–370. [CrossRef]
36. Wang, G.; Guo, L.; Wang, H.; Duan, H.; Liu, L.; Li, J. Incorporating mutation scheme into krill herd algorithm for global numerical optimization. *Neural Comput. Appl.* **2014**, *24*, 853–871. [CrossRef]

37. Wang, H.; Yi, J.-H. An improved optimization method based on krill herd and artificial bee colony with information exchange. *Memetic Comput.* **2017**. [CrossRef]
38. 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]
39. Wang, L.; Zheng, X.L.; Wang, S.Y. A novel binary fruit fly optimization algorithm for solving the multidimensional knapsack problem. *Knowl.-Based Syst.* **2013**, *48*, 17–23. [CrossRef]
40. Wang, G.-G.; Deb, S.; Coelho, L.D.S. Earthworm optimization algorithm: A bio-inspired metaheuristic algorithm for global optimization problems. *Int. J. Bio-Inspired Comput.* **2015**. [CrossRef]
41. Cui, Z.; Sun, B.; Wang, G.-G.; Xue, Y.; Chen, J. A novel oriented cuckoo search algorithm to improve DV-Hop performance for cyber-physical systems. *J. Parallel. Distr. Comput.* **2017**, *103*, 42–52. [CrossRef]
42. Wang, G.-G.; Gandomi, A.H.; Zhao, X.; Chu, H.E. Hybridizing harmony search algorithm with cuckoo search for global numerical optimization. *Soft Comput.* **2016**, *20*, 273–285. [CrossRef]
43. García, J.; Crawford, B.; Soto, R.; Castro, C.; Paredes, F. A k-means binarization framework applied to multidimensional knapsack problem. *Appl. Intell.* **2018**, *48*, 357–380. [CrossRef]
44. Deng, J.; Wang, L. A competitive memetic algorithm for multi-objective distributed permutation flow shop scheduling problem. *Swarm Evolut. Comput.* **2016**, *32*, 107–112. [CrossRef]
45. Wang, G.-G.; Tan, Y. Improving metaheuristic algorithms with information feedback models. *IEEE Trans. Cybern.* **2017**. [CrossRef] [PubMed]
46. Wang, G.-G.; Guo, L.; Gandomi, A.H.; Hao, G.-S.; Wang, H. Chaotic krill herd algorithm. *Inf. Sci.* **2014**, *274*, 17–34. [CrossRef]