

Article

List-Based Threshold Accepting Algorithm with Improved Neighbor Operator for 0–1 Knapsack Problem

Liangcheng Wu ^{1,2}, Kai Lin ^{1,2}, Xiaoyu Lin ¹ and Juan Lin ^{1,2,*}

¹ College of Computer and Information Science, Fujian Agriculture and Forestry University, Fuzhou 350002, China; 5221139006@fafu.edu.cn (L.W.); 12311093026@fafu.edu.cn (K.L.); xiaoyulin@fafu.edu.cn (X.L.)

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

* Correspondence: juan.lin@fafu.edu.cn

Abstract: The list-based threshold accepting (LBTA) algorithm is a sophisticated local search method that utilizes a threshold list to streamline the parameter tuning process in the traditional threshold accepting (TA) algorithm. This paper proposes an enhanced local search version of the LBTA algorithm specifically tailored for solving the 0–1 knapsack problem (0–1 KP). To maintain a dynamic threshold list, a feasible threshold updating strategy is designed to accept adaptive modifications during the search process. In addition, the algorithm incorporates an improved bit-flip operator designed to generate a neighboring solution with a controlled level of disturbance, thereby fostering exploration within the solution space. Each trial solution produced by this operator undergoes a repair phase using a hybrid greedy repair operator that incorporates both density-based and value-based add operator to facilitate optimization. The LBTA algorithm's performance was evaluated against several state-of-the-art metaheuristic approaches on a series of large-scale instances. The simulation results demonstrate that the LBTA algorithm outperforms or is competitive with other leading metaheuristics in the field.

Keywords: list-based; threshold accepting method; 0–1 knapsack problem; local search; hybrid greedy repair operator



Citation: Wu, L.; Lin, K.; Lin, X.; Lin, J. List-Based Threshold Accepting Algorithm with Improved Neighbor Operator for 0–1 Knapsack Problem. *Algorithms* **2024**, *17*, 478. <https://doi.org/10.3390/a17110478>

Academic Editor: Roberto Montemanni

Received: 7 September 2024
Revised: 17 October 2024
Accepted: 21 October 2024
Published: 25 October 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The list-based threshold accepting (LBTA) algorithm [1] falls under the category of threshold accepting (TA) algorithms [2]. In a manner analogous to simulated annealing (SA), LBTA accepts suboptimal solutions to prevent being trapped in local optima. Instead of setting to a fixed temperature, LBTA utilizes an adaptive threshold list that is updated iteratively based on the search space topology. This allows LBTA to achieve better results with fewer tuning parameters compared with SA and other local search methods. Owing to its flexibility, LBTA is effectively utilized in solving both combinatorial and continuous optimization problems, such as the vehicle routing problem (VRP) and its variants [3], the job shop scheduling problem (JSSP) [4], the zero-wait scheduling problem [5], and the traveling salesman problem (TSP) [6].

The 0–1 KP, which is recognized as a classic NP-hard problem in combinatorial optimization, has practical applications in various decision-making domains [7]. The problem involves a collection of n items, labeled from 1 to n ; each item has a weight w_i and a value v_i , as well as a maximum capacity c for the knapsack. The aim is to maximize the sum of the values of the items in the knapsack while making sure that the sum of the weights does not surpass the knapsack's capacity, as follows (see Equation (1)):

$$\begin{aligned} \max f(x) &= \sum_{i=1}^n v_i x_i \\ \text{s.t. } \sum_{i=1}^n w_i x_i &\leq c \wedge x_i \in \{0, 1\} \end{aligned} \quad (1)$$

where x_i represents the number of items i included into the knapsack, and $f(x)$ is the target function.

Aiming to fill the gap in the literature, this paper introduces a binary LBTA designed for tackling the 0–1 KP. The proposed algorithm features a feasible threshold list that effectively narrows the search space, alongside a hybrid greedy repair operator that surpasses traditional density-based operator regarding optimization and convergence. Moreover, a bit-flip mutation operator is employed to generate trial solutions, which provides an elastic local search area and a sufficient perturbation to escape local minimum. Extensive experimental evaluations demonstrate the efficacy and competitiveness of the LBTA algorithm, with comparative results on two large 0–1 KP instances highlighting its advantages.

The remainder of this paper is structured as follows: Section 2 offers a brief introduction of the basic LBTA algorithm, the 0–1 KP, and algorithms for the 0–1 KP. Section 3 presents the proposed LBTA algorithm, while Section 4 analyzes its behavior. In Section 5, the performance of the LBTA algorithm is compared with other state-of-the-art metaheuristics. Finally, Section 6 summarizes the findings of this study and outlines potential avenues for future research.

2. Related Work

2.1. List-Based Threshold Algorithm

LBTA is a random search algorithm that explores the solution space by employing a neighborhood search strategy. It allows the acceptance of suboptimal candidate solutions based on a predefined threshold value. As an extension of the TA metaheuristic, LBTA maintains a unique list of threshold values that decrease over the process of the iteration. This enables the algorithm to balance diversification and intensification without the need for an additional intervention or parameter configuration. The procedure consists of two main components: the initialization and updating of the threshold value list and a local search process.

2.1.1. Initialize the List of Threshold Values

During the first stage, an initial list of threshold values is generated through a particular local search method. The procedure then begins with the following steps: (i) selecting a neighboring solution \mathbf{s}' from a present solution \mathbf{s} by applying a local search move and (ii) calculating and normalizing the changing in the fitness function as follows:

$$T = |f(\mathbf{s}') - f(\mathbf{s})| / f(\mathbf{s}) \quad (2)$$

The newly generated threshold T is continually added to the list until it reaches its maximum length, which is controlled solely by the parameter L , representing the length of the list. In the initial phase, the local search procedure gathers information from the search space, while the normalization procedure helps to ensure a gradual decrease in the threshold values for the following iteration.

2.1.2. Iteration of the Algorithm

During the second stage, the iteration begins by generating a neighboring solution \mathbf{s}' from the initial randomized solution \mathbf{s} using a local search move. This moving strategy may be the same one adopted in the initialization, or a new method may be used. If the solution \mathbf{s}' is better than the original \mathbf{s} , then \mathbf{s} is updated with \mathbf{s}' , or a new threshold value T is generated using Equation (2). If T is less than the current largest threshold value T_{max} , \mathbf{s}' is accepted as the new solution, and T replaces T_{max} and enters the list. Otherwise, \mathbf{s}' is abandoned, and the algorithm proceeds with the original solution \mathbf{s} in the next iteration. This process is repeated until a stopping condition is satisfied.

2.1.3. Conservation of Threshold Strategy

In [1], to facilitate the ability to break free from local minima and enhance the chance of obtaining a better solution, this algorithm employs a strategy to save the threshold. This

involves keeping the values in the list sufficiently high by periodically updating the list with the maximum normalized threshold values discovered within a predefined number of feasible moves. This delay factor helps to slow down the adaptation of list values and avoid premature convergence to suboptimal solutions.

2.1.4. Pseudocode of LBTA

The pseudocode of LBTA is shown in Algorithm 1. We assume that the problem solved here is a maximization optimization problem that is consistent with the 0–1 KP. The algorithm is composed of two parts: initialization and iteration. In the initialization phase, a preliminary list of threshold values is established through a specific local search method, where L represents the length of the list. In the iteration phase, the algorithm generates a new solution by performing a local search move and updates the current solution and threshold list based on an acceptance criterion. The iteration continues until the maximum number of iterations G is achieved.

Algorithm 1: List-based threshold accepting algorithm

```

1 Initialize a feasible solution  $\mathbf{s}$ , a best solution  $\mathbf{s}_{\text{best}}$ ,  $T_{\text{max}} = +\infty$ , and an empty
  priority threshold list;
2 while the length of list is less than  $L$  do
3   | Generate a neighbor solution  $\mathbf{s}$ ;
4   | Use Equation (2) to generate threshold value  $T$ ;
5   | Insert  $T$  into the threshold list;
6 end
7 for  $g=1$  to  $G$  do
8   | Generate a neighbor solution  $\mathbf{s}'$ ;
9   | if  $f(\mathbf{s}') > f(\mathbf{s})$  then
10  |   | Update  $\mathbf{s}$  with  $\mathbf{s}'$ ;
11  |   | if  $f(\mathbf{s}) > f(\mathbf{s}_{\text{best}})$  then
12  |   |   | Update  $\mathbf{s}_{\text{best}}$  with  $\mathbf{s}$ ;
13  |   | end
14  | else
15  |   | Generate  $T$  with Equation (2);
16  |   | if  $T < T_{\text{max}}$  then
17  |   |   | Replace  $T_{\text{max}}$  with  $T$ ;
18  |   |   | Update  $\mathbf{s}$  with  $\mathbf{s}'$ ;
19  |   | end
20  | end
21 end

```

2.2. 0–1 KP

As a constrained optimization problem, dealing with the infeasible solution in the 0–1 KP requires an additional operator. Two the most commonly used methods in the literature are the penalty function and repair operation. The former involves designing a penalty function that assigns a lower fitness value to infeasible solutions, reducing their chance of being included in the next generation [8]. The repair operation is more widely used and typically involves a two-step greedy approach [9]. In step 1, when an infeasible solution is encountered, a greedy drop operator is applied to remove some items according to specific rules until the solution becomes feasible. In step 2, items are added back in until the solution becomes infeasible again. The most commonly used rule is based on profit density, which intuitively suggests that items with higher value and lower weight have a greater change in maximizing the total profits [10]. Algorithms 2 and 3 describe these steps respectively. In Algorithm 2, if the current solution is infeasible, the greedy drop operator checks every item in increasing order of v_i/w_i and changes x_i from 1 to 0

so that the lower density item can be removed at the very beginning. On the contrary, in Algorithm 3, the greedy add operator checks the items in descending order of v_i/w_i to include the higher profit density item in the first place.

Algorithm 2: Greedy drop operator

Input: An infeasible solution x , the sorted items H

Output: A new feasible solution x

```

1  $w = \sum_{i=1}^n w_i x_i$ ;
2 for  $i = n$  to 1 do
3   if  $x_{H[i]} = 1$  and  $w > c$  then
4      $x_{H[i]} = 0$ ;
5      $w = w - w_{H[i]}$ ;
6   end
7 end

```

Algorithm 3: Greedy add operator

Input: An feasible solution x , the sorted items H

Output: An new feasible solution x

```

1  $w = \sum_{i=1}^n w_i x_i$ ;
2 for  $i = 1$  to  $n$  do
3   if  $x_{H[i]} = 0$  and  $w \leq c$  then
4      $x_{H[i]} = 1$ ;
5      $w = w + w_{H[i]}$ ;
6   end
7 end

```

Early exact algorithms for the 0–1 KP, such as the dynamic programming approach [11], the branch and bound approach [12], and the enumeration approach [13], were designed for small-to-medium instances. As the size of instances increased, these methods failed to tackle the problem within an acceptable timeframe. Therefore, a wide range of metaheuristics have been put forward to tackle large-scale 0–1 KPs. Depending on the inspirational origins, these algorithms can be classified into four major categories: swarm-intelligence (SI)-based, bio-inspired, chemistry-based, and physics-based [14].

In the field of SI, traditional SI algorithms include the genetic algorithm (GA) [15], ant colony optimization (ACO) [16], and particle swarm optimization (PSO) [17]. Recently, a variety of novel SIs have proved to be very efficient in solving the 0–1 KP, such as the spider algorithm (BSSA), monkey algorithm (MA), whale optimization algorithm (WOA), cuckoo search (CS), chicken swarm optimization (CSO), wolf pack algorithm (WPA), and monarch butterfly optimization (MBO). Most of them design a binary version within the general algorithm framework, and use the aforementioned repair operator to fix the infeasible solution.

For example, a binary MBO (BMBO) [18] uses real-valued vectors and binary vectors to define search space and solution space separately. In chaotic MBO (CMBO) [19], 12 one-dimensional chaotic maps are utilized to adjust the migration procedures, and a Gaussian mutation operator is imposed on the worst solution to prevent premature convergence. A generalized opposition-based learning (OBL) MBO (OMBO) [20] employs OBL to speed up the convergence. Gaussian perturbation is also included to escape from a local optimum.

In the field of bio-inspired algorithms, a modified discrete shuffled frog leaping algorithm (MDSFLA) [21] includes a PSO-like technique and shuffled complex evolution to carry on local search. A binary version of differential evolution (BDE) [22] enriches the exploration and exploitation with four components: capability with the dual representation of solutions, an improved mapping method, enhanced mutation and one-point crossover, and a diversity mechanism.

In terms of chemistry-based methods, a binary chemical reaction optimization (CRO) framework is commonly applied for the 0–1 KP [8,23]. In the field of physics-based algorithms, a series of improved harmony search (HS) algorithms have emerged. A CS algorithm with global HS (CSGHS) [24] combines the exploration of global HS and the exploitation of CS to address the 0–1 KP. Good performance is obtained by comparing CS, SFLA, and DE in terms of search accuracy and convergence speed. A simplified binary HS (SBHS) [25] utilizes the distinctions among harmonies to generate new solutions and dynamically modifies the harmony memory consideration rate with the size of the dimension. An improved HS (IHS) [26] employs a feasible parameter-tuning way to generate new solution vectors so as to improve the correctness and convergence. A global-best HS (GHS) [27] adjusts the pitch-adjustment step through a global best harmony and adds a social dimension so that the solution can use the global best and local best information at the same time. Self-adaptive HS (SAHS) [28] uses harmony memory to automatically adjust parameters. A low-discrepancy sequence is exploited to the traditional pseudo-random initialization of the harmony memory. A list-based SA (LBSA) replaces the traditional temperature-based descent method with a practical list to manage the convergence pattern [29]. In our algorithm, we included a bit-flip mutation operator that generates trial solutions, which has been shown to offer an effective search space. Additionally, the noising method (NM) is another variant of SA. NM with six variants of NMs for the 0–1 KP has been proposed to address the 0–1 KP. These variants include two noise strategies, two noise variations of objective solutions, and two decreasing strategies [30]. Both LBSA and NM algorithms incorporate a hybrid greedy optimization operator that combines a density-based operator and a value-based operator, which has been proved to enhance algorithm performance. Therefore, we have also adopted this hybrid operator in our algorithm as a replacement for the traditional single optimization operator to improve the effectiveness of the algorithm.

Furthermore, in recent years, many novel algorithms have emerged to handle these problems, such as the reptile search algorithm (RSA) [31], binary slime mould algorithm (BSMA) [32], binary Archimedes optimization algorithm (BAOA), binary tunicate swarm algorithm (BTSA) [33], binary Harris Hawks optimization (BHHO) [34], binary marine predators algorithm (BMPA) [35], and binary elephant herding optimization (BinEHO) [36]. These algorithms have demonstrated significant potential in addressing complicated optimization problems due to their ability to effectively explore large search spaces and locate near-optimal solutions. We also include these algorithms and make a comparison with them in Section 5 to show the competitiveness of our proposed algorithm.

3. LBTA Algorithm with Enhanced Local Search for 0–1 KP

In this study, we introduce a novel version of the LBTA algorithm to tackle the 0–1 KP. The main framework can be summarized as follows: Initially, we redefine the solution representation for the 0–1 KP. Subsequently, we devise a rule for generating the threshold list, which adapts based on the evolving search landscape, thus enabling a more efficient exploration of the solution space. Following this, a bit-flip mutation operator is utilized to generate a diverse set of candidate solutions. These candidates are then refined using a hybrid greedy optimization operator, which employs both value-based and density-based strategies to repair and optimize the quality of the solutions.

3.1. Solution Representation

The solution for the 0–1 KP is represented by an object s with three attributes: $s.x$, $s.v$, and $s.w$. The n -bit binary array $s.x$ is generated randomly from $[0, 1]$, where $s.x_i = 1$ indicates that the corresponding item is included in the knapsack. $s.v$ represents the overall value of items in the knapsack, while $s.w$ represents the overall weight of the items in the knapsack.

3.2. The Initialization and Update of Threshold List

The threshold list is an essential part in the LBTA, which is responsible for providing a flexible search space. In the traditional LBTA, the list is generated through stochastic local

search, and the threshold values are normalized and stored in the list. In our proposed approach, we adopt a simpler and more efficient method: randomly selecting item values and inserting them into the list until the list is full, without the need to calculate fitness values.

To achieve a thorough and delicate search with a uniformly descending threshold list, we modify the basic framework of LBTA by utilizing a Markov chain to perform a series of local search, similar to SA. During each inner iteration, the decision to accept a worse solution is based on the current maximum value in the threshold list T_{max} . If the worst solution is adopted, the difference between the current solution and the candidate solution is accumulated, and the number of acceptance times is recorded. After completing the local search, we replace T_{max} with the average of cumulative differences. This approach provides a flexible threshold list that enables a border search space, occasional upward moves that promote escaping from local maximum points at the beginning, and a decreasing list of threshold values that lead to a thorough search and eventually reach the global maximum.

3.3. Improved Neighbor Operator

Apart from the threshold list, another crucial part of LBTA is the local search strategy. An effective strategy should be powerful enough to help the algorithm to break away from the attraction of the current solution while also being gentle enough to avoid becoming a stochastic local search. In particular, for the 0–1 KP, changing only one item at a time would result in a weak perturbation, leading to slow convergence, while changing too many items would result in a blind search. To address these issues, a feasible bit-flip mutation operator is proposed. During each local search, we randomly select items to include or exclude from the backpack. Two parameters, namely, FT and DT , are used to control the amount flipping and dropping times, respectively. Algorithm 4 outlines the process. Within the flipping number of FT , a random item index i is selected. If the current item has not been included yet, we flip $s.x_i$ from 0 to 1. Alternatively, if the current dropping number is less than DT , we exclude the item by setting $s.x_i$ from 1 to 0. The maximum flipping and dropping times are satisfied by iterating until the limits are reached.

Algorithm 4: A bit-flip mutation operator

Input: A solution s , flipping times FT , dropping times DT

Output: A new solution s

```

1   $ftTimes = 0, dtTimes = 0;$ 
2  while  $ftTimes < FT$  do
3      select an item  $i$  randomly;
4      if  $s.x_i == 0$  then
5           $s.x_i = 1;$ 
6           $s.w = s.w + w_i;$ 
7           $s.v = s.v + v_i;$ 
8           $ftTimes ++;$ 
9      else
10         if  $s.x_i == 1$  and  $dtTimes < DT$  then
11              $s.x_i = 0;$ 
12              $s.w = s.w - w_i;$ 
13              $s.v = s.v - v_i;$ 
14              $dtTimes ++;$ 
15         end
16          $ftTimes ++;$ 
17     end
18 end
19 return  $s$ 

```

3.4. Hybrid Greedy Repair and Optimized Operator

A traditional greedy repair operator only utilizes the profit density as the sole metric to determine the priority of item selection. One of the drawbacks of this approach is that it is challenging for high-value and high-weight items to be selected, which may lead to a local minimum. To solve this problem, we design a hybrid method that combines the greedy repair operator with an optimizing operator that considers both density and value. The hybrid approach involves two arrays, **HD** and **HV**, which keeps items organized in descending order based on the density of v_i/w_i and the value of v_i , respectively. During the first stage, we use only **HD** to drop items with lower density first. In the optimizing process, an additional parameter p is introduced to determine which array is used to invoke the greedy drop and greedy add operator (see Algorithm 5). The reason for dropping items using only **HD** is to preserve as much beneficial information as possible. If we choose the hybrid arrays, items possessing a higher value tend to be removed more often, and these items would need to be re-added during the optimization process, which would be a waste of iterations.

Algorithm 5: Hybrid greedy optimization operator

Input: A solution s , the probability p to select **HD** and **HV**
Output: A new feasible solution s

- 1 Use **HD** to call Algorithm 2 to repair s ;
- 2 Produce a random number $r \in [0, 1)$;
- 3 **if** $r < p$ **then**
- 4 | Use **HD** to call Algorithm 3 to optimize s ;
- 5 **else**
- 6 | Use **HV** to call Algorithm 3 to optimize s ;
- 7 **end**
- 8 **return** s

3.5. The Framework of the LBTA Algorithm for 0–1 KP

The framework of the LBTA algorithm is presented in Algorithm 6. Start with initializing the threshold list; a primary solution is generated randomly, repaired, and optimized using the hybrid greedy operator. In the inner iteration, a candidate solution s' is generated using the bit-flip mutation operator. If s' is better than s , or if the deviation between $s'.v$ and $s.v$ is less than $-T_{max}$, s is replaced with s' . After the number of local search iterations meets the Markov chain length, the current maximum threshold value is replaced with the average of the accumulation of deviation. The algorithm then starts a new local search until the global iteration limit is reached.

Algorithm 6: The framework of LBTA

Input: The number of generation G , Markov chain length M, L, p, FT , and DT
Output: A best solution s_{best}

```

1 for  $i = 1$  to  $L$  do
2   Randomly select an item  $i$ ;
3   insert the value of the item  $v_i$  into the list;
4 end
5 Randomly produce a solution  $s$ ;
6 Repair and optimize  $s$  with Algorithms 2 and 3;
7  $s_{best} \leftarrow s$ ;
8 for  $g = 1$  to  $G$  do
9    $sum = 0, count = 0$ ;
10  for  $m = 0$  to  $M$  do
11    Fetch  $T_{max}$  from the list;
12    Produce a candidate solution  $s'$  with Algorithm 4;
13    Repair and optimize  $s'$  with Algorithm 5;
14     $dif = s'.v - s.v$ ;
15    if  $dif > 0$  then
16       $s \leftarrow s'$ ;
17      if  $s_{best}.v > s.v$  then
18         $s_{best} \leftarrow s$ ;
19      end
20    else
21      if  $dif > -T_{max}$  then
22         $s \leftarrow s'$ ;
23         $sum = sum + dif$ ;
24         $count = count + 1$ ;
25      end
26    end
27  end
28  remove current  $T_{max}$  from the list, insert  $-sum/count$  into the list;
29 end
30 return  $s_{best}$ 

```

4. Behaviors Analysis

In this section, we present a series of experiments conducted to analyze the behavior of the LBTA algorithm. The experiments were performed on three sets of large 0–1 knapsack instances presented in [20]. These instances are classified by three types based on their features, as shown in Table 1. Each group consists of five instances of the 0–1 knapsack problem, along with the instance name; the number of items in each instance and the best solution for each instance are listed in Table 2. The optimal values are emphasized in bold. Instances KP01–KP05 represent uncorrelated cases, KP06–KP10 represent weakly correlated cases, and KP11–KP15 represent strongly correlated cases. The experiments were run on an Intel Core i7 CPU and an 8 GB RAM, using Java implementation, and each instance was run independently 50 times.

Table 1. Characteristics of three categories of large-scale 0–1 KP instances.

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

4.1. Parameters Settings

To validate the behavior of LBTA, we divided the parameters into two groups. The first group consisted of parameters for the basic LBTA framework, which included the number of generations G , the Markov chain length M , and the threshold list length L . The second group consisted of parameters for the local search, which included the probability p of the hybrid repair operator, the number of flipping times FT , and the number of dropping times DT . The total number of fitness evaluations was fixed at $G * M = 40,000$.

In the preliminary experiments, we determined the parameter values in the initial LBTA framework. For the other parameters, we set $p = 1$, $FT = 1$, and $DT = 1$ to ensure that these parameters did not cause interference. Specifically, we tried five values (50, 100, 200, 400, and 800) for G and ten values from 10 to 100 with 10 increments for L . We found that $G = 800$, $M = 50$, $L = 10$, and $p = 0.2$ were suitable for LBTA to achieve good performance.

4.2. Performance Tuning for the Feasible Bit-Flip Operator

To determine an appropriate combination for the number of bit flips, we conducted tests across a range of values for FT and DT . We recorded the number of functions that achieved a 100% success rate for each iteration, as most of these combinations were able to produce optimal results for the majority of instances. However, when $FT = 1$ and $DT = 1$, none of the functions were able to achieve the optimal result with a 100% success rate. Therefore, we adjusted the values to set FT from 10 to 80 with 10 increments and DT from 0 to 5 with 1 increment.

When an item is excluded from the knapsack, it still has the chance to be re-included. However, if DT is the same value as FT , numerous items may be repeatedly included and excluded, leading to unnecessary iteration. Thus, we aim to fill up the knapsack as quickly as possible while also introducing local disturbance. This is why we set DT to be much lower than FT .

The histograms in Figure 1 depict the simulation results for the different combinations of FT and DT . From these histograms, we obtained the following results: (1) For FT , there is an insignificant difference between 15 functions. When FT equals 40, 50, 70, and 80, 14 of 15 functions achieve a 100% success rate, while for FT equal to 30 and 60, 13 of 15 functions obtain a 100% success rate. For the remaining values, 12 of 15 functions reach a 100% success rate. (2) For DT , there is a significant difference between six values. When $DT = 1$, all FT values were able to produce the best results. (3) In general, the value of FT should not be too large to avoid random search. Based on the simulation results, we selected $FT = 40$, $DT = 1$ for the following simulations.

Table 2. Comparison of the LBTA algorithm on the first set of 15 instances.

Instance	BKV	HGGA (2020)				NM (2021)				CMBO (2018)				LBTA			
		Best	Mean	Worst	SD	Best	Mean	Worst	SD	Best	Mean	Worst	SD	Best	Mean	Worst	SD
KP01	40,686	40,686	40,685.00	40,685	6.00×10^{-2}	40,685	40,684.88	40,684	2.20×10^{-1}	40,686	40,683.00	40,683	7.10×10^{-1}	40,686	40,685.07	40,685	2.46×10^{-3}
KP02	50,592	50,592	50,592.00	50,592	0.00	50,592	50,592.00	50,592	0.00	50,592	50,590.00	50,590	4.90×10^{-1}	50,592	50,592.00	50,592	0.00
KP03	61,846	61,846	61,845.90	61,845	3.70×10^{-1}	61,846	61,845.32	61,845	4.40×10^{-1}	61,845	61,841.00	61,840	1.38	61,846	61,845.33	61,845	1.62×10^{-3}
KP04	77,033	77,033	77,033.00	77,033	0.00	77,033	77,032.92	77,032	1.50×10^{-1}	77,033	77,031.00	77,031	3.10×10^{-1}	77,033	77,032.77	77,032	1.30×10^{-3}
KP05	102,316	102,316	102,315.99	102,3165	2.00×10^{-2}	102,316	102,316.00	102,316	0.00	102,316	102,314.00	102,313	1.11	102,316	102,315.53	102,314	1.96×10^{-3}
KP06	35,069	35,069	35,069.00	35,069	0.00	35,069	35,069.00	35,069	0.00	35,069	35,067	35,064	1.47	35,069	35,069.00	35,069	0.00
KP07	43,786	43,786	43,786.00	43,786	0.00	43,786	43,785.96	43,785	8.00×10^{-2}	43,786	43,784.00	43,781	1.34	43,786	43,786.00	43,786	0.00
KP08	53,553	53,553	53,552.48	53,552	5.00×10^{-1}	53,553	53,552.02	53,552	4.00×10^{-2}	53,552	53,552.00	53,552	0.00	53,553	53,552.10	53,552	1.87×10^{-3}
KP09	65,710	65,710	65,709.11	65,709	2.00×10^{-1}	65,709	65,709.00	65,709	0.00	65,710	65,709.00	65,708	5.80×10^{-1}	65,710	65,709.13	65,709	1.52×10^{-3}
KP10	118,200	118,200	118,200.00	118,200	0.00	108,200	108,200.00	108,200	0.00	108,200	108,200.00	108,200	0.00	118,200	118,200.00	118,200	0.00
KP11	40,167	40,167	40,167.00	40,167	0.00	40,167	40,167.00	40,167	0.00	40,167	40,167.00	40,166	1.40×10^{-1}	40,167	40,167.00	40,167	0.00
KP12	49,443	49,443	49,443.00	49,443	0.00	49,443	49,443.00	49,443	0.00	49,433	49,432.00	49,433	2.49	49,443	49,443.00	49,443	0.00
KP13	60,640	60,640	60,640.00	60,640	0.00	60,640	60,640.00	60,640	0.00	60,640	60,640.00	60,639	1.40×10^{-1}	60,640	60,640.00	60,640	0.00
KP14	74,932	74,932	74,932.00	74,932	0.00	74,932	74,932.00	74,932	0.00	74,932	74,932.00	74,931	2.70×10^{-1}	74,932	74,932.00	74,932	0.00
KP15	99,683	99,683	99,683.00	99,683	0.00	99,683	99,683.00	99,683	0.00	99,683	99,682.00	99,672	2.23	99,683	99,683.00	99,683	0.00
order		15	12	14	10	13	9	12	10	13	1	2	2	15	11	13	11
rank		1				3				3				1			

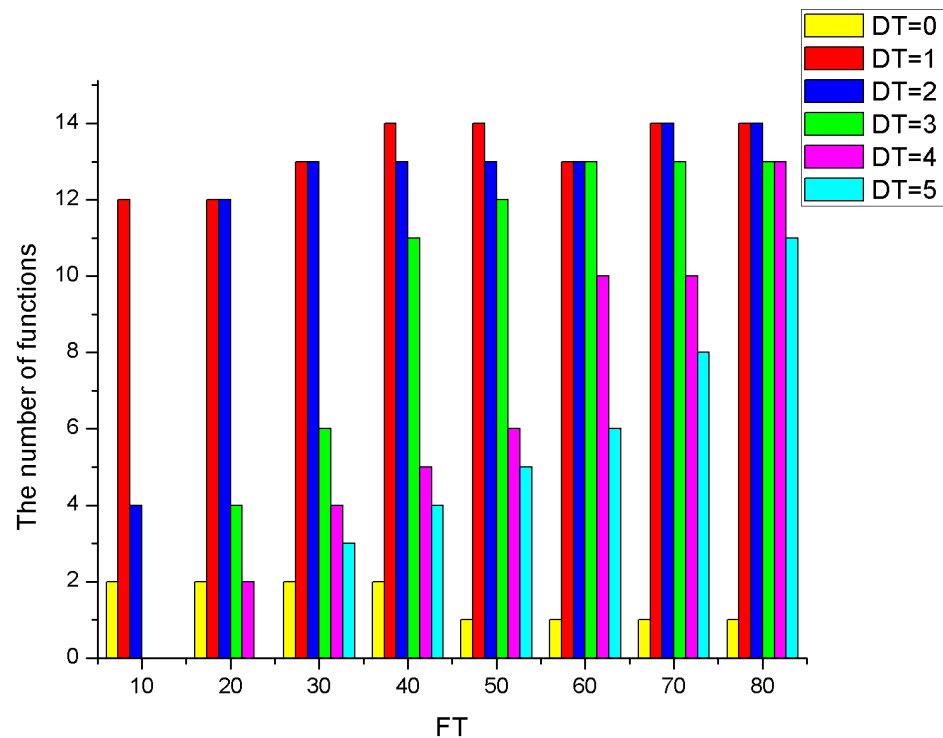


Figure 1. Performance comparison of LBTA with the different numbers of bit-flip operators.

5. Competitiveness of the LBTA Algorithm

To access the competitive performance of the LBTA algorithm, we reviewed a broad range of references and selected two widely used benchmark sets from recent studies. The first set comprises three types of 0–1 KPs, as described in the previous section. The second set, more frequently used in recent references, was sourced from [25] and included instances with sizes ranging from 100 to 6400. The results were directly obtained from the original literature. In all experiments, the maximum number of generations was set to 40,000, and each instance was evaluated using 50 independent experiments. The performances of various algorithms were assessed using the best, worst, average, and standard deviation (SD) of the solution values. The optimal values are emphasized in bold.

5.1. Competitiveness of the LBTA Algorithm on the First Set of Instances

In this subsection, the performance of LBTA was evaluated against three state-of-the-art metaheuristics: CMBO algorithm [19], NMs [30], and HGGA [37]. The comparisons are presented in Table 2. The results show that HGGA and LBTA outperform NMs and CMBO. Regarding the best solution, the two algorithms yield comparable results. NMs failed to obtain the best solution for KP01 and KP09, while CMBO did not find the best solution for KP03 and KP08. Only HGGA and LBTA were successful in obtaining the best solution for every instance. Regarding the worst, mean solution, and SD, HGGA and LBTA also outperformed the other two algorithms.

5.2. Competitiveness of the LBTA Algorithm on the Second Set of Instances

First, we made a comparison on 25 small instances where the size varied between 8 and 24. Details of these 25 KPs were obtained from David Pisinger's research [38]. To evaluate the effectiveness and accuracy of LBTA, it is assessed in relation to recent studies, including BEO [35], BinEHO [36], BMPA [39], BinRSA [31], BSMA [32], BHHO [34], BTSA [33], and BAOA [40]. In Table 3, the average results and gap values of each algorithms are provided. The proposed method demonstrates superior or equal performance compared with other algorithms across all 25 problems, as indicated by the gap results. Notably, LBTA outperforms all competing algorithms, achieving the best solution in 100% of the instances.

While algorithms such as BMPA, BSMA, and BEO also obtain the best solution in 100% of the 24 instances, their performance falls slightly short of LBSA on specific problems—BMPA and BEO on the P24 instance and BSMA on the P9 instance. Additionally, although BinRSA achieves the best average in 20 instances, and BinEHO, BHHO, BTSA, and BAOA secure the best results in 17 out of 25 instances, none match the consistent and optimal performance of LBSA across all test cases.

Furthermore, the LBTA algorithm was evaluated on large-scale instances from David Pisinger's optimization codes [41]. The datasets used in this study fall into three different categories: datasets 1–6 are high dimensional uncorrelated, datasets 7–12 are high dimensional weakly correlated, and datasets 13–18 are high dimensional strongly correlated. In Table 4, a comparison of the performance of the LBTA algorithm with that of IGA-SA [42], BinRSA [31], BSMA [32], BTSA [33], BAOA [40], and BHHO [34] is presented. The results, as shown in Table 4, demonstrate that LBSA performs similarly as or better than other algorithms in 14 out of 18 problems according to the mean results, matching or exceeding the number of optimal solutions obtained by the other algorithms. While the BinRSA algorithm achieves the same number of optimal solutions, our proposed LBTA ranks first overall when all algorithms are considered, with BinRSA ranking third behind BSMA. The results clearly show that LBTA performs only slightly worse than BinRSA in the KP1_5000, KP2_5000, and KP3_5000 instances, and its performance in the KP1_2000 instance is marginally worse than those of BSMA, BAOA, and BHHO. However, LBTA outperforms all algorithms in the median items for the KP2_500 instance and matches the best value achieved by other algorithms in 13 other instances.

In conclusion, the LBTA algorithm demonstrated strong performance across multiple large-scale instances, particularly on high-dimensional weakly correlated and strongly correlated datasets, showcasing its robust competitiveness. Although it performed slightly less effectively than certain algorithms on a few specific instances, it ranked first overall, proving the effectiveness and robustness of the LBTA algorithm in solving 0–1 KPs.

6. Conclusions

This paper presents an enhanced local search algorithm, LBTA, tailored to effectively address the 0–1 knapsack problem (KP). The LBTA algorithm employs several key mechanisms to improve the performance in both solution quality and computational efficiency. By incorporating a feasible threshold list, this algorithm is able to regulate convergence speed, ensuring that the search process balances between intensification and diversification. The integration of a bit-flip mutation operator further strengthens the exploration capabilities, enabling a more comprehensive search for the solution space and reducing the likelihood of premature convergence. Moreover, the algorithm incorporates a hybrid repair operator that merges value-based and density-based greedy strategies. This hybrid operator enhances the quality of candidate solutions by guiding the search towards more optimal areas of the solution space. Through this combination of strategies, LBTA demonstrates its capability to provide high-quality solutions for the 0–1 knapsack problem while maintaining simplicity in its implementation.

The LBTA algorithm's performance has been systematically tested on two large instances of the 0–1 KP, and its results have been compared with those of several cutting-edge metaheuristics found in the existing literature. The outcomes of these simulations reveal that LBTA is not only straightforward to implement but also exhibits remarkable efficiency in solving the 0–1 knapsack problem. A limitation observed in the initial testing instances indicates that the performance of LBTA is slightly less stable compared with those of other algorithms, which may be attributed to the random flip mutation operator used during the neighborhood search. While this operation introduces search diversity, it may also lead to a degree of blind searching. Consequently, in our next steps, we plan to integrate all operators with a status feedback mechanism. This approach will allow for the adaptive selection of operators, each serving distinct purposes, based on the current evolutionary status, thereby facilitating a more balanced and effective search. Additionally, we aim to expand the LBTA algorithm to address more complicated and practical real-world optimization problems, further enhancing its applicability and effectiveness in various domains.

Author Contributions: J.L. and L.W. conceived and designed the experiments; K.L. performed the experiments; J.L. and K.L. analyzed the data; X.L. contributed analysis tools; L.W. wrote the paper. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Nature Science Foundation of Fujian Province of China (No. 2023J01078).

Data Availability Statement: Our research data are shared in <https://github.com/LiangchengWu1/> (accessed on 17 October 2024).

Conflicts of Interest: Liangchen Wu, Kai Lin, Xiaoyu Lin, Juan Lin declare no conflict of interest. The founding sponsors had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, and in the decision to publish the results.

References

1. Tarantilis, C.D.; Kiranoudis, C.T. A list-based threshold accepting method for job shop scheduling problems. *Int. J. Prod. Econ.* **2002**, *77*, 159–171. [\[CrossRef\]](#)
2. Dueck, G.; Scheuer, T. Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *J. Comput. Phys.* **1990**, *90*, 161–175. [\[CrossRef\]](#)
3. Tarantilis, C.D.; Ioannou, G.; Kiranoudis, C.T.; Prastacos, G.P. Solving the open vehicle routing problem via a single parameter metaheuristic algorithm. *J. Oper. Res. Soc.* **2005**, *56*, 588–596. [\[CrossRef\]](#)
4. Lee, D.S.; Vassiliadis, V.S.; Park, J.M. A novel threshold accepting meta-heuristic for the job-shop scheduling problem. *Comput. Oper. Res.* **2004**, *31*, 2199–2213.
5. Lee, D.S.; Vassiliadis, V.S.; Park, J.M. List-based threshold-accepting algorithm for zero-wait scheduling of multiproduct batch plants. *Ind. Eng. Chem. Res.* **2002**, *41*, 6579–6588. [\[CrossRef\]](#)
6. Ilhan, I.; Gökmen, G. A list-based simulated annealing algorithm with crossover operator for the traveling salesman problem. *Neural Comput. Appl.* **2022**, *34*, 7627–7652. [\[CrossRef\]](#)
7. Cho, M. The knapsack problem and its applications to the cargo loading problem. *Anal. Appl. Math.* **2019**, *48*.

8. Truong, T.K.; Li, K.; Xu, Y. Chemical reaction optimization with greedy strategy for the 0-1 knapsack problem. *Appl. Soft Comput. J.* **2013**, *13*, 1774–1780. [[CrossRef](#)]
9. Chu, P.C.; Beasley, J.E. A genetic algorithm for the multidimensional knapsack problem. *J. Heurist.* **1998**, *4*, 63–86. [[CrossRef](#)]
10. Dantzig, G.B. Discrete-variable extremum problems. *Oper. Res.* **1957**, *5*, 266–288. [[CrossRef](#)]
11. Martello, S.; Pisinger, D.; Toth, P. Dynamic programming and strong bounds for the 0-1 knapsack problem. *Manag. Sci.* **1999**, *45*, 414–424. [[CrossRef](#)]
12. Kolesar, P.J. A branch and bound algorithm for the knapsack problem. *Manag. Sci.* **1967**, *13*, 723–735. [[CrossRef](#)]
13. Cabot, A.V. An enumeration algorithm for knapsack problems. *Oper. Res.* **1970**, *18*, 306–311. [[CrossRef](#)]
14. Fister, I., Jr.; Yang, X.S.; Fister, I.; Brest, J.; Fister, D. A brief review of nature-inspired algorithms for optimization. *arXiv* **2013**, arXiv:1307.4186.
15. Yadav, R.K.; Gupta, H.; Jhingran, H.; Agarwal, A.; Gupta, A. An Enhanced Genetic Algorithm to Solve 0/1 Knapsack Problem. *Int. J. Comput. Sci. Inf. Secur. (IJCSIS)* **2017**, *15*, 150–154.
16. Alzaqebah, A.; Abu-Shareha, A.A. Ant Colony System Algorithm with Dynamic Pheromone Updating for 0/1 Knapsack Problem. *Int. J. Intell. Syst. Appl.* **2019**, *11*, 9. [[CrossRef](#)]
17. Nguyen, P.; Wang, D.; Truong, T.K. A new hybrid particle swarm optimization and greedy for 0–1 knapsack problem. *Indones. Electr. Eng. Comput. Sci.* **2016**, *1*, 411–418. [[CrossRef](#)]
18. Feng, Y.; Wang, G.G.; Deb, S.; Lu, M.; Zhao, X.J. Solving 0-1 knapsack problem by a novel binary monarch butterfly optimization. *Neural Comput. Appl.* **2015**, *28*, 1619–1634. [[CrossRef](#)]
19. 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.* **2016**, *10*, 135–150. [[CrossRef](#)]
20. Feng, Y.; Wang, G.G.; Dong, J.; Wang, L. Opposition-based learning monarch butterfly optimization with Gaussian perturbation for large-scale 0-1 knapsack problem. *Comput. Electr. Eng.* **2018**, *67*, 454–468. [[CrossRef](#)]
21. Bhattacharjee, K.K.; Sarmah, S.P. Shuffled frog leaping algorithm and its application to 0/1 knapsack problem. *Appl. Soft Comput.* **2014**, *19*, 252–263. [[CrossRef](#)]
22. Ali, I.M.; Essam, D.; Kasmari, K. An efficient differential evolution algorithm for solving 0–1 knapsack problems. In Proceedings of the 2018 IEEE Congress on Evolutionary Computation (CEC), Rio de Janeiro, Brazil, 8–13 July 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 1–8.
23. Truong, T.K.; Li, K.; Xu, Y.; Ouyang, A.; Nguyen, T.T. Solving 0-1 knapsack problem by artificial chemical reaction optimization algorithm with a greedy strategy. *J. Intell. Fuzzy Syst.* **2015**, *28*, 2179–2186. [[CrossRef](#)]
24. Feng, Y.; Wang, G.G.; Gao, X.Z. A Novel Hybrid Cuckoo Search Algorithm with Global Harmony Search for 0-1 Knapsack Problems. *Int. J. Comput. Intell. Syst.* **2016**, *9*, 1174–1190. [[CrossRef](#)]
25. 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](#)]
26. Mahdavi, M.; Fesanghary, M.; Damangir, E. An improved harmony search algorithm for solving optimization problems. *Appl. Math. Comput.* **2007**, *188*, 1567–1579. [[CrossRef](#)]
27. Omran, M.G.; Mahdavi, M. Global-best harmony search. *Appl. Math. Comput.* **2008**, *198*, 643–656. [[CrossRef](#)]
28. Wang, C.M.; Huang, Y.F. Self-adaptive harmony search algorithm for optimization. *Expert Syst. Appl.* **2010**, *37*, 2826–2837. [[CrossRef](#)]
29. Zhan, S.H.; Zhang, Z.J.; Wang, L.J.; Zhong, Y.W. List-Based Simulated Annealing Algorithm with Hybrid Greedy Repair and Optimization Operator for 0-1 Knapsack Problem. *IEEE Access* **2018**, *6*, 54447–54458. [[CrossRef](#)]
30. Zhan, S.; Wang, L.; Zhang, Z.; Zhong, Y. Noising methods with hybrid greedy repair operator for 0–1 knapsack problem. *Memetic Comput.* **2019**, *12*, 37–50. [[CrossRef](#)]
31. Ervural, B.; Hakli, H. A binary reptile search algorithm based on transfer functions with a new stochastic repair method for 0–1 knapsack problems. *Comput. Ind. Eng.* **2023**, *178*, 109080. [[CrossRef](#)]
32. Abdel-Basset, M.; Mohamed, R.; Sallam, K.M.; Chakraborty, 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](#)]
33. Kaur, S.; Awasthi, L.K.; Sangal, A.L.; Dhiman, G. Tunicate Swarm Algorithm: A new bio-inspired based metaheuristic paradigm for global optimization. *Eng. Appl. Artif. Intell.* **2020**, *90*, 103541. [[CrossRef](#)]
34. Heidari, A.A.; Mirjalili, S.; Faris, H.; Aljarah, I.; Mafarja, M.; Chen, H. Harris hawks optimization: Algorithm and applications. *Future Gener. Comput. Syst.* **2019**, *97*, 849–872. [[CrossRef](#)]
35. Abdel-Basset, M.; Mohamed, R.; Mirjalili, S. A binary equilibrium optimization algorithm for 0–1 knapsack problems. *Comput. Ind. Eng.* **2021**, *151*, 106946. [[CrossRef](#)]
36. Hakli, H. BinEHO: A new binary variant based on elephant herding optimization algorithm. *Neural Comput. Appl.* **2020**, *32*, 16971–16991. [[CrossRef](#)]
37. Chen, Z.; Zhong, Y.; Lin, J. Hybrid greedy Genetic Algorithm for solving 0-1 knapsack problem. *J. Comput. Appl.* **2021**, *41*, 87.
38. Pisinger, D. Where are the hard knapsack problems? *Comput. Oper. Res.* **2005**, *32*, 2271–2284.
39. Abdel-Basset, M.; Mohamed, R.; Chakraborty, R.K.; Ryan, M.; Mirjalili, S. New binary marine predators optimization algorithms for 0–1 knapsack problems. *Comput. Ind. Eng.* **2021**, *151*, 106949. [[CrossRef](#)]

40. Hashim, F.A.; Hussain, K.; Houssein, E.H.; Mabrouk, M.S.; Al-Atabany, W. Archimedes optimization algorithm: A new metaheuristic algorithm for solving optimization problems. *Appl. Intell.* **2021**, *51*, 1531–1551. [[CrossRef](#)]
41. Pisinger, D. Instances of 0/1 Knapsack Problem. 2018. Available online: <https://github.com/likr/kplib> (accessed on 17 October 2024).
42. Ezugwu, A.E.; Pillay, V.; Hirasen, D.; Sivanarain, K.; Govender, M. A comparative study of meta-heuristic optimization algorithms for 0–1 knapsack problem: Some initial results. *IEEE Access* **2019**, *7*, 43979–44001. [[CrossRef](#)]

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