An Efficient Heuristic Algorithm for Solving 0-1 Knapsack Problem

Hong Yang*, Xiong Guo

Internet of Things Research Centre, China Electronics Standardization Institute, Beijing 100007, China

*Corresponding author: yanghong@cesi.cn

Abstract. The model for 0-1 knapsack problem based on greedy strategy and expectation efficiency has some advantages, e.g., quick convergence and low complexity, but it also inherently exists some limitations to be enhanced and improved, such as the restricted greedy and the overmuch calculation on expectation efficiency. To this end, an efficient algorithm is proposed to optimize the previous model. At first, the greedy degree is maximized, that is, some items are permanently loaded into knapsack in advance as many as possible. Then, the conventional expectation efficiency model is improved greatly, at the same time, the number of calculations with respect to expectation efficiency is decreased by ignoring some unnecessary items. Compared to the previous algorithm, the experimental results reveal that the proposed optimization algorithm has the significant advantages in time.

Keywords: 0-1 Knapsack Problem, Greedy Strategy, Expectation Efficiency, Optimization Model, Calculation Times.

1. Introduction

Knapsack Problem (KP) belongs to the classical NP-hard problem. From the perspective of type, KP is composed of multiple KP, complete KP and 0-1 KP (KP01). In particular, KP01 is the most fundamental and the most common KP. At the same time, multiple KP and complete KP can be solved by being translated into KP01 [1]. In addition, from the perspective of objectives, KP consists of single-objective KP, bi-objective KP and multiple-objective KP. Similarly, both single-objective KP and bi-objective KP can be addressed by translating them into single-objective KP. Given this, the single-objective KP plays the important role in the fields of academia and industry.

In the recent years, there have been a number of proposals for solving KP01, including the exact methods and the heuristic methods. To be specific, the solution got by the former is usually the optimal while that got by the latter usually and only approaches the optimal (i.e., the approximate solution). To the best of our knowledge, the traditional and exact ones mainly include dynamic programming, branch and bound, and backtracking. Although some new exact ones have been proposed, they are always subject to dynamic programming method, branch and bound method, or backtracking method. Here, we emphasize that it is very difficult for these traditional and exact ones to show good performance in terms of the large-scale scenarios. To this end, the heuristic and approximate ones have been developed, mainly including two aspects, i.e., meta-heuristic algorithm and mathematic-based heuristic algorithm.
Specifically, the meta-heuristic algorithm (such as bee optimization algorithm [2], butterfly optimization algorithm [3], bat optimization algorithm [4], whale optimization algorithm [5] and frog optimization algorithm [6]) is explored by introducing the inherent survival laws in terms of the natural environments. However, the meta-heuristic algorithm need many times iterations to achieve desired result, which is difficult to be accepted in practice. By contrast, the pure mathematical heuristic solution focuses on the artificial mathematical modeling to design a simple algorithm, which has the advantages of fast convergence and low complexity on the basis of guaranteeing the optimal solution. Therefore, researchers pay more attention to this model.

Among many heuristic methods, the solution based on the expectation efficiency model is highly practical. It is developed by improving the economic pre-estimation, and its basic theory was proposed by Tian et al. in 2013 [7]. Later, relevant scholars [8-13] further optimize this model. However, all of the optimizations follow two laws: First, they load some objects into the backpack in advance according to greedy strategy. Second, they calculate the expectation efficiency by the expectation efficiency model for each remaining object. It is no doubt that those two laws make the result convergence to optimal result but there are still a lot of space to improve. First, the degree of greed is limited, that is, loading objects without maximizing greed. Second, calculation of expectation efficiency is too much. It is unreasonable to calculate the expected efficiency for all the remaining objects. Given this, based on these two improvement spaces, we continue to study the heuristic method based on expectation efficiency to solve the 0-1 knapsack problem.

2. Instructions of Origin Algorithm

2.1. 0-1 Knapsack Problem

Given n objects, 0-1 knapsack problem can be formulated as:

\[
\begin{align*}
\text{Maximize:} & \quad \text{optp} = \sum_{i=1}^{n} p_i x_i \\
\text{Subject to:} & \quad \sum_{i=1}^{n} w_i x_i \leq M
\end{align*}
\]  

where \(w_i\) and \(p_i\) are weight and value of object \(i\) respectively. \(M\) is the weight capacity of the knapsack. \(\text{optp}\) is the target that needs to be optimize. \(x_i\) whose value is 0 or 1 denotes whether the object \(i\) was loaded into the knapsack, \(x_i\) is equal to 1 represents that object \(i\) is not load into knapsack, otherwise object \(i\) cannot be loaded into knapsack. In brief, the ultimate goal of 0-1 knapsack problem is to find an optimal combine \(W = \{w_1, w_2, \cdots, w_n\}, P = \{p_1, p_2, \cdots, p_n\}, X = \{x_1, x_2, \cdots, x_n\}\) to maximize the \(\text{optp}\).

2.2. Description on Greedy Degree

The design of greed degree depends on two phases: First, objects are sorted according to the value/weight. Second, the first \(m\) objects are loaded into knapsack. Thus, the key lies in the determination of \(m\): If \(m\) is too small and not greedy enough, it will lead to the increase of expectation efficiency calculation in the later period (performance problem). If \(m\) is too large, excessive greed will lead to deviation from the optimal solution (correctness problem). The current greed degree design [10-13] mainly relies on two types of constraint conditions, as shown in formula (2) and formula (3) respectively:

\[
\frac{1}{M} \sum_{i=1}^{m} w_i \leq \rho
\]

Where \(\rho\) is between 0 and 1.
\[
\frac{1}{GW} \sum_{i=1}^{m} w_i \land \frac{1}{Goptp} \sum_{i=1}^{m} p_i \leq \lambda
\]
\[
\frac{1}{GW} \sum_{i=1}^{m+1} w_i \land \frac{1}{Goptp} \sum_{i=1}^{m+1} p_i > \lambda
\]
\[
\frac{1}{GW} \left( \sum_{i=1}^{\frac{m}{2}} w_i + \sum_{i=1}^{m} p_i \right) \land \frac{1}{Goptp} \left( \sum_{i=1}^{\frac{m}{2}} p_i + \sum_{i=1}^{m} p_i \right) < \xi
\]
\[
\frac{1}{GW} \left( \sum_{i=1}^{\frac{m}{2}} w_i + \sum_{i=1}^{m+1} p_i \right) \land \frac{1}{Goptp} \left( \sum_{i=1}^{\frac{m}{2}} p_i + \sum_{i=1}^{m+1} p_i \right) \geq \xi
\]

Where \( \lambda \) and \( \xi \) are between 0 and 1 and \( \lambda > \xi \). \( GW \) and \( Goptp \) are total weight and total value of the objects which are selected by greedy algorithm.

2.3. Description on Expectation Efficiency

The expectation efficiency model mainly refers to the dynamic expectation efficiency model, whose mathematical expression is shown as follows.

\[
f_i = \frac{r_i}{r'_{i-1}} \times \frac{r_{i-1} \left( M - \sum_{k=1}^{m} w_k - \sum_{k=m+1}^{i-1} w_k x_k \right) - (n-i+1)p_i}{M - \sum_{k=1}^{m} w_k - \sum_{k=m+1}^{i-1} w_k x_k - (n-i+1)w_i}
\]

There are two parts in this model. The first part is the ratio of the cost performance of the current object to the cost performance of the previous object, indicating the expectation ratio. The second part is the expectation which the backpack can hold the remaining objects. In particular, the domain of formulate (4) is \([m + 1, n]\), so the expectation efficiency model needs to be executed \( n - m \) times.

There are two shortcomings of expectation efficiency: First, the calculation is complicated. Second, it is a bit rigid and wasteful to have to calculate the expectation efficiency of every remaining object.

3. The Proposed Algorithm

In this section, system optimization is made based on the shortcomings of the original algorithm (lack of greed, high expectation efficiency and complex model), including two parts: greed level optimization and expectation efficiency optimization.

3.1. Optimization of Greedy Degree

At the initial stage, set the qualification conditions as shown as follows.

\[
\begin{align*}
\sum_{i=1}^{\theta} w_i & \leq M \\
\sum_{i=1}^{\theta+1} w_i & > M
\end{align*}
\]

It is obvious that this design guarantees maximum greed. However, test with a large number of examples shows that for some special examples, the object \( \theta \) cannot be included in the optimal solution. Thus, to correct formulate (5), we set

\[
\Delta = M - \sum_{i=1}^{\theta} w_i - w_k \geq 0
\]

Where \( k \) is between \( \theta + 1 \) and \( n \). Assume the \( k \)-th object has the largest weight, if

\[
\frac{w_k}{M - \sum_{i=1}^{\theta} w_i} \geq \chi
\]

Then formula (5) is still right and \( m = \theta \). Otherwise, object \( \theta \) is removed from knapsack and \( m = \theta - 1 \).

According to the above description, the optimization algorithm of greedy degree is as follows.

01. Calculate \( \theta \) by formulate (5);
02. for object $\theta + 1$ to object $n$, do
03. if satisfy formulate (6), then
04. pick out this object;
05. endif
07. Select object which has maximum weight;
08. if satisfy formulate (7), then
09. $m = \theta$;
10. else
11. $m = \theta - 1$;
12. endif

3.2. Optimization of Greedy Degree

3.2.1. Computation Optimization. As mentioned above, it is not necessary to calculate the expectation efficiency of all the remaining objects, because the first objects have been locked. If we calculate the expectation efficiency of some remaining objects, it will only increase the time cost. If

$$w_k > M - \sum_{i=1}^{\theta} w_i$$

(8)

It means that the object is not likely to be loaded into the backpack, that is, there is no need to carry out subsequent expectation efficiency calculations on it. Assuming the number of objects satisfy formula (7) is $\psi$, then the expectation efficiency calculation times after optimization is $n - m - \psi$.

3.2.2. Model Optimization. In fact, formula (4) does not necessarily highlight the expectation ratio. Instead, the overall expectation can be evaluated as to whether the object in the backpack is commensurate with its corresponding value, and if so, how much value can be obtained later based on the same expectation. Denote $t$ as the rate of value of the whole thing after the backpack is loaded into the next object, and we have:

$$t_{ri} = \frac{\sum_{k=1}^m p_k + \sum_{k=m+1}^i p_k x_k}{\sum_{k=1}^m w_k + \sum_{k=m+1}^i w_k x_k}$$

(9)

We calculate expectation efficiency values of remaining objects according to formula (9) and then calculate the expectation efficiency according to the formulate (10):

$$f_i = t_{r_{i-1}} \left( M - \sum_{k=1}^m w_k - \sum_{k=m+1}^{i-1} w_k x_k \right) - (n - i + 1) p_i$$

$$M - \sum_{k=1}^m w_k - \sum_{k=m+1}^{i-1} w_k x_k - (n - i + 1) w_i$$

(10)

Compared to formulate (4), the improved expectation efficiency model is relatively simple. In particular, the removed object when calculate $t_{ri}$ is correspond to the remove object when calculate $M - \sum_{k=1}^m w_k - \sum_{k=m+1}^{i-1} w_k x_k$.

According all, the optimization algorithm of expectation efficiency is as follows.

01. for object $\theta + 1$ to object $n$, do
02. if satisfy (8), then
03. Ignore this object;
04. endif
05. endfor
06. $n = n - m - \psi$;
07. for object $\theta + 1$ to object $n$, do
08. if $M - \sum_{k=1}^m w_k - \sum_{k=m+1} w_k x_k < 0$, then
09. Find the object with the minimum expectation efficiency value;
10. endif
11. \( t r_i \) is calculated according to formula (9);
12. \( f_i \) is calculated according to formula (10);
13. endfor
14. for object \( \theta + 1 \) to object \( n \), do
15. Find objects with higher expectation efficiency in turn and loading them into knapsack;
16. endfor

4. Simulation Results
The simulation experiment consists of two parts. The first part gives an example to illustrate the feasibility of the algorithm proposed in this paper, and gives detailed calculation steps. The second part is a longitudinal comparison experiment based on several classic schemes and the literature [13]. Specifically, the algorithm in this paper and the literature [13] are written in C++. The hardware configuration is Intel(R) Xeon e5-2680, 2.49 GHZ CPU, and 7.99 GB RAM. The parameter \( \chi \) in this paper is set to 0.95.

4.1. Page Numbers
Examples: \( n = 16, M = 750, W = \{120, 98, 94, 118, 77, 70, 90, 115, 110, 73, 81, 106, 113, 80, 87, 2\}, P = \{240, 192, 184, 229, 149, 135, 173, 221, 210, 139, 154, 201, 214, 150, 163, 3\} \). The calculation steps are as follows.

Step1: According to Eq. (5), we get \( \sum_{k=1}^{7} w_k = 667 \) and \( \sum_{k=1}^{10} w_k = 782 \), which indicates that 750 is between 667 and 782 when \( \theta = 7 \);

Step2: We iterate through object 8 to object 16, and we found that only object 11 satisfies the condition according to Eq. (6);

Step3: According to Eq. (7), the weight of the backpack is 750-667 = 83 and 81/83 = 0.9759 > 0.9, then \( m = \theta = 7 \);

Step4: According to Eq. (8), only objects 10, 11, 14 and 16 need to be calculated for expectation efficiency. At this time, Then the subsequent objects are 8, 9, 10 and 11 as \( n = 7 + 4 = 11 \);

Step5: The first expectation efficiency is calculated according to Eq. (9) and Eq. (10):

\[
750 - 667 > 0, \\
tr_7 = \frac{1302}{667} = 1.9520,
\]

\[
f_6 = \frac{1.9520(750-667)-(11-8+1)+139}{(750-667)-(11-8+1)+73} = 1.8851; \]

Step6: The second expectation efficiency is calculated according to Eq. (9) and Eq. (10):

\[
750 - 667 - 73 > 0, \\
tr_8 = \frac{1302+139}{667+73} = 1.9473,
\]

\[
f_9 = \frac{1.9473(83-73)-(11-9+1)+154}{(83-73)-(11-9+1)+81} = 1.8993; \]

Step7: The third expectation efficiency is calculated according to Eq. (9) and Eq. (10):

\[
83 - 73 - 81 < 0, \text{ and we select object 8 as } f_8 < f_9,
\]

\[
tr_9 = \frac{1302+154}{667+81} = 1.9465,
\]

\[
f_{10} = \frac{1.9465(10-8+73)-(11-10+1)+150}{(10-8+73)-(11-10+1)+80} = 1.8741; \]

Step8: The fourth expectation efficiency is calculated according to Eq. (9) and Eq. (10):

\[
10 - 81 + 73 - 80 < 0, \text{ and we select object 10 due to } f_7,
\]

\[
tr_{10} = \frac{1302+154+3}{667+81+2} = 1.9453,
\]

\[
f_{11} = \frac{1.9453(2-80+80)-(11-11+1)+3}{(2-80+80)-(11-11+1)+2} = +\infty; \]

Step9: Object 11 and object 9 are loaded into the knapsack in order, and the optimal solution is \{1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1\}. The optimal target is 1459, and its change process is shown in Fig. 1.
Fig 1. The changing process of objective function value.

As shown in Fig. 1, the algorithm in this paper has gone through 13 iterations. Objects 1 to 7 are loaded into the knapsack in the first seven iterations. The next four iterations have no change in the objective value, meaning that the expectation efficiency values are calculated. Object 16 and 9 are sequentially loaded into the knapsack in the last two iterations. However, the algorithm in [13] requires 19 iterations. Object 1 to 6 to be loaded into the knapsack in the first six iterations. The next ten iterations have no change in the objective value, because this process calculates the expectation efficiency values. Object 7, 9 and 16 are sequentially loaded into the knapsack in the last three iterations.

4.2. Comparison Analysis
In order to further illustrate the efficiency of the proposed optimization algorithm, 10 classic examples in [6] and [13] are selected for comparison experiments. The experimental metrics are the optimal objective value, the number of iterations, and the execution time. The results are as follows.

Table 1 shows that both algorithms can obtain the optimal objective value, and the optimization algorithm proposed in this paper has distinct improvements in the number of iterations and execution time. The lower number of iterations is due to the increased number of objects loaded into the knapsack in advance and the reduced number of calculations of expectation efficiency. In fact, the execution time mainly includes the cost-effective sorting time, early loading time, expectation efficiency calculation time, and expectation efficiency sorting time. The second part accounts for a small part, and the first and third parts account for a large amount. This explains some phenomenon of execution time: (1) the number of iterations of the same algorithm changes, and its execution time does not change much; (2) the number of iterations of different algorithms is similar, and the execution time is different. In comparison, the algorithm in this paper has slightly advantages.
Table 1. The comparison testing results.

| No. | Algorithm       | The best value | The number of iterations | Difference | Running time/ms |
|-----|----------------|----------------|--------------------------|------------|-----------------|
| 1   | This paper     | 295            | 9                        | 5          | 9.7229          |
|     | Reference-13   | 295            | 14                       |            | 11.5141         |
| 2   | This paper     | 1024           | 17                       | 7          | 10.3809         |
|     | Reference-13   | 1024           | 24                       |            | 11.9597         |
| 3   | This paper     | 35             | 3                        | 3          | 8.5236          |
|     | Reference-13   | 35             | 6                        |            | 11.0438         |
| 4   | This paper     | 23             | 3                        | 2          | 8.4984          |
|     | Reference-13   | 23             | 5                        |            | 10.3617         |
| 5   | This paper     | 481.0694       | 12                       | 7          | 9.7941          |
|     | Reference-13   | 481.0694       | 19                       |            | 11.9346         |
| 6   | This paper     | 52             | 5                        | 8          | 8.8477          |
|     | Reference-13   | 52             | 13                       |            | 11.4982         |
| 7   | This paper     | 107            | 7                        | 2          | 9.2436          |
|     | Reference-13   | 107            | 9                        |            | 11.3035         |
| 8   | This paper     | 9767           | 16                       | 13         | 10.1258         |
|     | Reference-13   | 9767           | 29                       |            | 14.2671         |
| 9   | This paper     | 130            | 4                        | 2          | 8.7311          |
|     | Reference-13   | 130            | 6                        |            | 11.0972         |
| 10  | This paper     | 1025           | 17                       | 8          | 10.4263         |
|     | Reference-13   | 1025           | 25                       |            | 12.2034         |

5. Conclusions

KP01 pays an important role in the field of combinational optimization problem, and the method based on expectation efficiency has some distinguished advantages, such as fast convergence and low complexity. This paper further improves the expectation efficiency model from two aspects. At first, to optimize the greedy degree to the most extent, that is, more objects are put into knapsack in advance. Then, the expectation efficiency model is simplified in terms of computation and running times. Compared to the latest work, the proposed optimization algorithm performs smaller iterations and lower running time.

Acknowledgments

This work was financially supported by the project <Test and Verification of Internet of things Technology based on Edge Computing>, which is one of the “Industry and Information Public Service Capacity Improvement Project in 2019”.

References

[1] Buhrman, H., Loff, B., Torenvliet, L.: Hardness of approximation for knapsack problems. Theory of Computing Systems 56, 372-393 (2015).

[2] Cao, J., Yin, B., Lu, X., et al.: A modified artificial bee colony approach for the 0-1 knapsack problem. Applied Intelligence 48(6), 1582-1595 (2018).

[3] Feng, Y., Wang, K., Li, W., et al.: Multi-strategy monarch butterfly optimization algorithm for discounted 0/1 knapsack problem. Neural Computing and Applications 30(10), 3019-3036 (2018).

[4] Huang, X., Li, P., Pu, Y: Amplitude angle modulated bat algorithm with application to zero-one knapsack problem. IEEE Access 7, 27957-27969 (2019).

[5] Abdel-Basset, M., El-Shahat, D., Sangaiah, A.K.: A modified nature inspired meta-heuristic whale optimization algorithm for solving 0-1 knapsack problem. International Journal of Machine Learning and Cybernetics 10(3), 495-514 (2019).
[6] Bhattacharjee, K.K., Sarmah, S.P.: Shuffled frog leaping algorithm and its application to 0/1 knapsack problem. Applied Soft Computing 19, 252-263 (2014).
[7] Tian, Y., Lv, J., Zheng, L.: An algorithm of 0-1 knapsack problem based on economic model. Journal of Applied Mathematics and Physics 1(10), 31-35 (2013).
[8] Shi, L., Zhang, Y., Lv, J.: Optimization algorithm of 0-1 knapsack problem based on absolute greedy and expected efficiency. Application Research of Computers 31(3), 684-687 (2014).
[9] Shi, L., Lv, J.: New optimization algorithm of 0-1 knapsack problem. Application Research of Computers 31(4), 997-1000 (2014).
[10] Zhang, L., Zhang, H.: Solution of 0-1 knapsack problem based on expected efficiency and linear fitting. Journal of Computer Applications 34(9), 2581-2584 (2014).
[11] Su, A., Gao, W., Zhang, Y.: Solving 0-1 knapsack problem by greedy degree and improved expectation efficiency model based on region limitation. Application Research of Computers 32(11), 3304-3308 (2015).
[12] Lv, J., Wang, X., Huang, M., et al.: Solving 0-1 knapsack problem by greedy degree and expectation efficiency. Applied Soft Computing 41, 94-103 (2016).
[13] Zhang, L., Lv, J.: A heuristic algorithm based on expectation efficiency for 0-1 knapsack problem. International Journal of Innovative Computing, Information and Control 14(5), 1833-1854 (2018).