Proceedings of the 3rd International Conference on Signal Processing and Machine Learning
DOI: 10.54254/2755-2721/5/20230666

Comparison of dynamic programming and greedy algorithms and the way to solve 0-1 knapsack problem

Yitian Wu

School of Computer Science, Central South University for Nationalities, Wuhan, Hubei Province, China, 430000

wuyitian2022@163.com

Abstract. The 0-1 knapsack problem is widely used in reality and it belongs to NP-hard problems. Starting from the basic ideas and time complexity of the algorithm, this paper analyzes how to choose a strategy to solve the problem. This paper aims to solve practical problems and analyzes the 0-1 knapsack problem in combination with the real-life cargo delivery problem. Dynamic programming and greedy algorithms are used to tackle the problem respectively, and the advantages and disadvantages of two strategies are discussed, so as to analyze how to decide which strategy to adopt to solve the problem when encountering the 0-1 knapsack problem in an actual situation. In the face of large-scale problems, this paper suggests choosing greedy algorithm because it will save a lot of time. In the face of small-scale problems that require absolute solutions, this paper suggests choosing dynamic programming to solve the problem.

Keywords: 0-1 Knapsack Problem, Dynamic Programming, Greedy Algorithm, Maximal Knapsack Packing, Optimal Solution.

1. Introduction

The knapsack problem was proposed by Merkel and Heilmann in 1978. Later, by studying its characteristics, it is shown that this problem is a typical NP-complete problem. It is widely used in various industrial situations, such as capital budget, cargo handling, and storage allocation problems, which can be transformed into the 0-1 knapsack problem, so the study of the solution of the 0-1 knapsack problem has important practical significance [1]. At present, the existing solution methods can be divided into two categories: the exact algorithm, such as dynamic programming algorithms, backtracking method, and branch and bound method. The other is approximate algorithms, such as greedy, ant colony algorithm and genetic algorithm. This paper mainly selects a typical algorithm for analysis from two types of algorithms, namely dynamic programming algorithm and greedy algorithm [2]. In the early 1950s, the American mathematician R. Behlman and others proposed the famous optimization principle when studying the optimization problem of a multi-stage decision-making process, and thus established dynamic programming. Dynamic programming has a wide range of applications, including engineering technology, the economy, industrial production, military and automation control, and other fields. Greedy algorithm was first proposed by J. C. Wnsdorff in 1823, which means that the current optimal choice is always made when solving a problem, that is, the local optimal solution. Greedy algorithms have two basic elements: greedy selection and optimal substructure. It is the closest to the human daily thinking of a problem-solving strategy and is essentially an improved

DOI: 10.54254/2755-2721/5/20230666

hierarchical processing method. Although it does not guarantee that the solution is the best choice but can determine the feasible range for the problem, it uses the top-down way to make a choice by iterative method, which compared with other algorithms, has a speed advantage [3]. This paper mainly discusses and analyzes the complexity of these two algorithms, the basic idea analysis and the actual cargo transportation problem. In addition, the advantages and disadvantages of the two algorithms are also studied. The study of 0-1 knapsack problem can provide examples for people to solve practical problems. At the same time, the research and comparison of two common algorithms can help people make better decisions when they encounter this kind of problem. In addition, this paper will help those who study algorithms better understand the 0-1 knapsack problem as well as greedy algorithms and dynamic programming.

2. Description of the knapsack problem

Text description: There are n items and a backpack. The weight of item i is w_i, its value is v_i and the knapsack capacity is C. The purpose of this question is to maximize the value of the items packed into the backpack. In this process, there are only two options to put in and not put in, and it is not possible to choose a part to put in.

Mathematical description: Given C > 0, w_i> 0, v_i> 0, 1 ≤ i ≤ n, find a n-element 0-1 vector (x1, x2,… x_n), xi ∈{0, 1}, 1 ≤ i ≤ n, such that$\sum_{i=1}^{n} w_i x_i \leq$ C.

3. The strategy for finding the absolute optimal solution — dynamic programming method

The 0-1 knapsack problem requires that the items loaded into the knapsack have the greatest value. This is an optimal solution problem. To solve the absolute optimal solution, the dynamic programming algorithm is a very good strategy.

3.1. Basic idea

Core idea of this algorithm is to gradually expand the optimal value of the original problem through the optimal value of its smallest sub-problems and solve them one by one, that is, from the bottom to the top. During this process, some auxiliary data is retained to construct the optimal solution. From the core idea, it can be seen that the problem that can be tackled by this algorithm must contain two elements: the characteristics of overlapping subproblem and optimal substructure [4]. The optimal substructure property is that the optimal solution of the original problem contains the optimal solution of the subproblem. Overlapping subproblems are well understood, i.e., a smaller scale subproblem can be many subproblems of a larger scale problem. If without this property, the subproblems are independent of each other, then it is better to solve it with a divide-and-conquer strategy.

3.2. Dynamic programming method to solve the 0-1 knapsack problem

3.2.1. Establish mathematical recursion relation. The optimal value of the sub-problem constructs m(i, j): it is the maximum value of the items that can be loaded into the knapsack when the capacity of the knapsack is j, and the optional items are i, i+1, ... n. When solving m(i, j), the subproblem m(i+1, j) smaller than it has been solved to get the optimal value. For the i-th item currently considered, there are two cases: 1) the current backpack capacity j cannot hold the item i, and its optimal solution is equivalent to m (i+1, j); 2) the current knapsack capacity j can hold item i. At this point, the goods can be loaded or not loaded. If it is loaded into the backpack, the current value will be added to the optimal value of the previous sub-question (m (i+1, jwi)). Therefore, if the current capacity can hold item i, there will be two values. What this problem requires is the maximum value of the current backpack, so choose the larger of the two values [5].

The mathematical recursion can be expressed as follows:

$$\begin{cases} \max\{m(i+1), m(i+1, j-w_i)+v_i\} & j \geq w_i \\ m(i+1, j) & 0 \leq j < w_i \end{cases} \quad (1)$$

DOI: 10.54254/2755-2721/5/20230666

3.2.2. Time complexity. As can be seen from the above description of the main algorithm, the time complexity is O(nC) computation time. When capacity c of the knapsack is large, the algorithm takes more time. For example, when C > 2n, the time complexity becomes O(n2n). If the knapsack capacity is large enough, this paper suggests abandoning the dynamic programming algorithm in favor of the greedy algorithm.

4. Strategy for finding approximate optimal solution - greedy algorithm

The greedy algorithm considers only the current optimal choice rather than the overall optimal choice each time. Of course, to get the optimal solution, at this time, this problem should be proved to have the property of greedy choice and the property of optimal substructure. Although not all problems can be solved by greedy method, but for many problems, when the scale is large enough, the result calculated by it is a good approximate solution of the optimal solution.

4.1. Basic idea

Choosing the optimal greedy strategy is the first problem to be considered when using the greedy method to solve the problem. The optimal greedy strategy plays an important part in making the solution closer to optimal solution. There are three strategies:

The first is the maximum value greedy strategy: If the backpack capacity allows, select the most valuable items to load until the backpack capacity is insufficient.

The second is the minimum weight greedy strategy: in the case of backpack capacity permits, select the smallest weight items into the backpack, until the backpack can not pack the rest of the items.

The third is the value-to-weight greedy strategy: in the case of backpack capacity permits, select the items with the largest v_i/w_i value to load, until the back packaging is not enough to carry the remaining items.

With strategy 1, if the most valuable item is too heavy, then the backpack capacity will not be used effectively. Choosing strategy 2, if the weight of the items is also low, then it is difficult to guarantee the maximum value of the items in the backpack. Strategy 3 considers both value and weight. Intuitively, optimal solution can be obtained according to this strategy. In this paper, greedy strategy 3 is chosen as the optimal greedy strategy to design the algorithm. The steps of how to solve the problem with greedy strategy 3 are as follows.

Find the value weight ratio of a given item v_i/w_i (i = 1, 2, ... , n). Sort items in non-increasing order of value. Repeat following steps until the conditions are not met: put the item which has the highest value to weight ratio into the backpack, calculate the remaining space in the backpack. If the current backpack remaining capacity can be loaded into the item, then load. Otherwise, select the next item.

4.2. Using greedy algorithm to solve the 0-1 knapsack problem

When the weight of the backpack and the items it contains are not of the same weight, such as the capacity of the backpack is much greater than the weight of the item to be selected, or when the backpack is replaced by a truck, the greedy algorithm is an excellent method. According to the basic idea of the greedy method, the first thing to do is to sort items in descending order of unit weight value. Then select and load items in descending order of unit weight value until they no longer fit [6].

4.3. Time complexity

When greedy algorithm solves the problem, the time complexity of sorting is O(n$\log n$) and the time consumption of greedy selection is O(n). Therefore, the time complexity is O(n$\log n$).

5. Cargo loading

With the development of science and technology, container loading robots have been put into use. In this process, the robot needs to transport the container to the ship. The weight of the ship is fixed, and all that is to be done is to load as much cargo as possible without exceeding the carrying capacity of the ship (assuming that the volume of the cargo is not limited). Since the container is indivisible, there are

DOI: 10.54254/2755-2721/5/20230666

only two options: loading or unloading the container, which is a typical 0-1 knapsack problem [7].

5.1. Example introduction

Suppose there is a small boat that can carry 10 tons of cargo and there are 5 pieces of cargo that need to be loaded on board. The weights of these five cargoes are 8 tons, 4 tons, 3 tons, 4 tons, and 2 tons, respectively. The values of these five cargoes are 7, 5, 5, 3, and 2, respectively. The specific weight and value are shown in Table 1. The corresponding importance levels are 2, 5, 5, 3, 2. In this case, the people carrying the cargo need to maximize the importance of the cargo as much as possible without exceeding the carrying capacity of the ship.

Table 1. The specific weight and value.

Number	Weight	Value
1	8	7
2	4	5
3	3	5
4	4	3
5	2	2

5.2. Dynamic programming

It is suggested that the maximum cargo load when loading a 0-ton container should be calculated firstly, then the cases with a 1-ton container, a 2-ton container, and a 3-ton container orderly, and so on. Therefore, an array dp[i][j] is defined in this paper, where i indicates the serial number of the container, j indicates the current ship's loading, dp[i][j] indicates the maximum value of the i-th container, and the ship's loading is j. When both i and j loop to the last value, dp[i][j] is the maximum value sought. Take one of these five containers in order, and first judge whether they can be loaded. If the container is loaded beyond the maximum capacity of the ship, it will not be loaded. At this time, the maximum value of the cargo loaded by the ship is the maximum value of the cargo loaded after the previous container is loaded. If the cargo can be loaded, it is then judged whether the value of the loaded cargo after loading the container is greater or the value of the loaded cargo when the container is not loaded is greater [8]. Go through them one by one, record the maximum value of the cargo on board, and fill in the form. The example is shown in Table 2.

Table 2. The details of dynamic programming.

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	7	7	7
2	0	0	0	0	5	5	5	5	7	7	7
3	0	0	0	5	5	5	5	10	10	10	10
4	0	0	0	5	5	5	5	10	10	10	10
5	0	0	2	5	7	7	7	10	10	12	12

5.2.1. Analysis. In this case, the complexity of the algorithm is O(10N). The author found that in this example, because the size of the problem was small, it didn't take much time. Therefore, when faced

DOI: 10.54254/2755-2721/5/20230666

with a small-scale problem to be tackled, dynamic programming is a good method to find the optimal solution by dynamic programming.

5.2.2. Greedy algorithm. When using the greedy algorithm, the first thing to do is to calculate the unit values of the five containers, as shown in Table 3.

Table 3. The unit value of the goods.

Item	Unit value
1	0.875
2	1.250
3	1.667
4	0.75
5	1

After obtaining the unit values of the five containers, the five containers are sorted in descending order of unit values: [3, 2, 5, 1, 4].

5.3. Analysis

The main computation time of greedy algorithm will be spent on the non-increasing sorting of items according to the value proportion. If quick sorting is used to realize the value proportion sorting, the time complexity of this algorithm is O(n$\log n$) [9].

6. Discussion

In the real world, the carrying capacity of cargo ships is thousands or even hundreds or thousands of tons, and the number of containers is also hundreds or thousands. At this point, the time complexity of the dynamic program may become O(n^2). However, the time complexity of the greedy algorithm is O(n$\log n$) [10]. Also, there may be more than one cargo ship at a port, which can make the workload very heavy. So at this time, the greedy algorithm can save a lot of time cost. Although the greedy algorithm is only an approximate solution, it can help us save a lot of time [10].

7. Conclusion

There are many strategies to solve the 0-1 knapsack problem. It is important to choose the suitable strategy to solve the problem. In general, when the absolute optimal value is needed, the dynamic programming method is recommended. If the capacity of the backpack is much higher than the weight of the item, this paper suggests a greedy method to find an approximate solution that is close to the optimal solution. When facing different problems, it is necessary to analyze the specific problem and find the best solution that can meet the conditions and improve efficiency. There are many ways to solve the problem. The author only introduces two methods: dynamic programming and greedy algorithm. Neither of these methods can really solve the 0-1 knapsack problem, because both methods have their own shortcomings. In the future research, the author will conduct a deeper study on the algorithm and come up with some better algorithms to solve life's problems. For 0-1 knapsack problem, the author will study more algorithms, summarize their respective advantages and disadvantages to get a more suitable solution.

Acknowledgment

First, I want to thank my professors and teachers because of their help and answers to my doubts from the topic selection to the final draft of the paper. I finished this paper under their guidance. In addition, the support of my parents and friends is also indispensable. Without their help, I could not have finished my thesis.

DOI: 10.54254/2755-2721/5/20230666

References

[1] Martello, S., & Toth, P. (1990). Knapsack problems: Algorithms and computer implementations. Chichester: John Wiley & Sons.
[2] Bellman, R. (1957). Dynamic programming. Princeton University Press.
[3] Boyar, J., Epstein, L., Favrholdt, L. M., Kohrt, J. S., Larsen, K. S., Pedersen, M. M., et al. (2006). The maximum resource bin packing problem.Theoretical Computer Science, 362(1–3), 127–139.
[4] Martello, S., & Toth, P. (1990). Knapsack problems: Algorithms and computer implementations. Chichester: John Wiley & Sons.
[5] Glover, F. (2013). Advanced greedy algorithms and surrogate constraint methods for linear and quadratic knapsack and covering problems. European Journal of Operational Research, 230(2), 212–225.
[6] Kellerer, H., Pferschy, U., & Pisinger, D. (2004). Knapsack problems. Berlin: Springer.
[7] Martello, S., Pisinger, D., & Toth, P. (1999). Dynamic programming and strong bounds for the 0-1 knapsack problem. Management Science, 45, 414–424.
[8] Pisinger, D. (1995). A minimal algorithm for the multiple-choice knapsack problem. European Journal of Operational Research, 83(2), 394–410.
[9] Pisinger, D. (2007). The quadratic knapsack problem – a survey. Discrete Applied Mathematics, 155(5), 623–648.
[10] Gai, L., & Zhang, G. (2009). Hardness of lazy packing and covering. Operations Research Letters, 37(2), 89–92.