International Journal of Multidisciplinary Research and Development
Online ISSN: 2349-4182, Print ISSN: 2349-5979; Impact Factor: RJIF 5.72
Received: 15-07-2020; Accepted: 30-07-2020; Published: 16-08-2020
www.allsubjectjournal.com
Volume 7; Issue 8; 2020; Page No. 147-151

The comparison between the dynamic programming method and integer programming method for knapsack problem

Xintong Yang
School of Mathematics and Statistics, Northeastern University at Qinhuangdao, Qinhuangdao, China

Abstract
The knapsack problem is an important problem in operations research. By taking some practical instances with the use of the dynamic programming method and integer programming method to solve the knapsack problem, this paper makes a comprehensive comparison of the two methods, and the conclusion is: When the integer programming method is applied to solve the knapsack problem, it is necessary to decide to use implicit enumeration method or branch and bound method according to the type of the knapsack problem. While the dynamic programming method pays more attention to the middle process of putting items.

Keywords: Knapsack problem, Dynamic programming, Integer programming, MATLAB

1. Introduction

Since the knapsack problem was first proposed by Merkle and Hellman in 1978[1], it has been applied to practical problems such as investment decision-making, so it quickly became a focus studied in the field of operations research[2-9]. There are different methods to solve this problem, such as dynamic programming method, integer programming method, genetic algorithm and ant colony algorithm, in which the integer programming method and dynamic programming method are most commonly used[10-19]. What are the characteristics of the the two methods? And what kind of problems are these two methods more suitable for solving? If these two problems can be answered, we will formulate problem-solving strategies more quickly and more efficiently, and better solve the problem. Therefore, this article intends to make a comparison and analysis of these two methods to get more valuable conclusions.

2. The integer programming method to solve knapsack problem

2.1 Mathematical model of 0-1 knapsack problem

The 0-1 knapsack problem is described as follows: It is known that there is a knapsack with the maximum capacity a kg. There are *n* kinds of items that can be put into it, and each item can be put one at most. The weight of the *i* th item is a_i kg, and the value of the *i* th item is c_i. Then how to arrange the items to maximize the total value in the knapsack?

According to the description above, in 0-1 knapsack problem, the plan for each item is to be put or not, which is the only constraint. Therefore, the mathematical model of 0-1 knapsack problem can be expressed as follows:

$$x=\begin{cases}1 & \text{put the } i\text{th item in the knapsack}\\ 0 & \text{do not put the } i\text{th item in the knapsack}\end{cases} \quad (1)$$

$$\max\ z=\sum_{i=1}^{n}c_i x_i$$

$$s.t.\begin{cases}\sum_{i=1}^{n}a_i x_i \le a & (i=1,2\cdots,n)\\ x_i=0\ \text{or}1\end{cases}, \quad (2)$$

According to the implicit enumeration method which is commonly used to solve 0-1 integer programming problems, the specific method is described as follows:

Step 1: If there are *n* independent variables, there are 2^n values of independent variables.

First, all the possible values of independent variables are listed;

Step 2: Calculate the value of the objective function *z* under the value of the first set of independent variables ;

Step 3: Test whether the values of the first set of independent variables meet the constraint conditions. If they all meet the constraint conditions, then this set is taken as an alternative; if the constraint conditions are not met, this set of independent variable is discarded;

Step 4: If the values of the first set of independent variables meet all constraints, then a filter condition can be formed: The *z* calculated is the lower bound of the final objective function. If the following *z* calculated later is less than it, then the following set of independent variables should be discarded.

After the values of all the listed set of independent variables following by the above four steps, the maximum value of the objective function is the lower bound finally obtained of the objective function, and the values of corresponding independent variables are the optimal solution. The calculation process is shown in Table 1:

Table 1: The process of implicit enumeration method

$(x_1, x_2, \cdots, x_n)$	**The value *of z***	**Constraint conditions** *a b ⋯ p*	**Filter condition**
(0,0,⋯,0)	z_1	√ √⋯	$z \geq z_1$
(0,0,⋯,1)	z_2	…	…
…	…	…	…

In order to elaborate the above processes, we give an example to illustrate next.

Example 1:
A truck with a maximum capacity of 10 tons is used to carry three kinds of goods (each kind of goods is at most one). As for each goods, the weight of per unit and its value are shown in Table 2. Then which goods should be put to maximize the total value in the knapsack?

Table 2: The weight and value of three goods

Number of the goods(*i*)	1	2	3
Weight of per unit /t	3	4	5
Value of per unit(c_i)	4	5	6

Solution:
According to the description of the problem above, we establish a 0-1 linear programming model: Supposing that x_i represents whether the *i* th kind of goods are put in or not, and *z* represents the total value of the goods:

$$x = \begin{cases} 1 & \text{put } i\text{th item in the bag} \\ 0 & \text{do not put } i\text{th item in the bag} \end{cases} \tag{3}$$

$$\max z = 4x_1 + 5x_2 + 6x_3$$
$$s.t. \begin{cases} 3x_1 + 4x_2 + 5x_3 \leq 10 \\ x_i = 0 \text{ or } 1 \end{cases} \quad (i = 1, 2, \cdots, n) \tag{4}$$

By using the implicit enumeration method we obtain the optimal solution of this problem. The process is shown in Table 3:

Table 3: Solving example 1 with the implicit enumeration method

$(x_1, x_2, \cdots, x_n)$	**The value *of z***	**Constraint**	**Filter condition**
(0,0,0)	0	√	$z \geq 0$
(0,0,1)	6	√	$z \geq 6$
(0,1,0)	5	--	--
(1,0,0)	4	--	--
(0,1,1)	11	√	$z \geq 11$
(1,0,1)	10	--	--
(1,1,0)	9	--	--
(1,1,1)	15	×	--

It can be seen from the above table, the optimal solution and the maximum value of the objective function are $x_1 = 0, x_2 = 1, x_3 = 1$, $\max z = 11$, which means that when the truck load the 2th and 3th goods the total value reaches its maximum 11.

2.2 The pure integer programming model for knapsack problem

For the general knapsack problem, the variable x_i represents the quantity of the *i* th item in the knapsack, which is also the only one constraint. Therefore, the mathematical model can be expressed as:

$$\max z = \sum_{i=1}^{n} c_i x_i$$
$$s.t. \begin{cases} \sum_{i=1}^{n} a_i x_i \leq a \\ x_i \geq 0 \text{ and } x_i \text{ is an integer} \end{cases} \quad (i = 1, 2 \cdots, n) \tag{5}$$

Assuming that the original pure integer linear programming problem is *A* and the corresponding relaxation problem is *B*. The branch and bound method can be described as follows:
Step 1: Write the relaxation problem of the original problem *B*;
Step 2: Calculate the optimal solution and objective function value of problem *B*. If it has no feasible solution, then there is no feasible solution to the original problem *A* neither. If the optimal solution is an integer solution, then it is also the optimal solution of the original problem *A*. If the optimal solution of problem *B* is not an integer solution, then we perform the following steps;
Step 3 (Branching process): Pick one variable that do not meet the requirement of integer to branch, and divide it into two branches

$$x_j \leq \lfloor \bar{b}_j \rfloor_j, \quad x_j \geq \lfloor \bar{b}_j \rfloor + 1 \tag{6}$$

This two conditions should be added to the original problem respectively, which causes two subsequent questions *C*, *D*;
Step 4 (Bounding process): If the optimal solutions of the two subsequent problems calculated are both integer solutions, we shill compare the values of the objective function of the two subsequent problems, and the solution with the larger value of the objective function is the optimal solution. If one of the solutions of the subsequent problem is not integer, then we should first compare the value of the objective function of the two subsequent problems, and then choose the one with larger objective function value to branch again and get two subsequent problems marked as *E*, *F*.
Step 5: Calculate the optimal value and objective function value of the problem *E*, *F*. If the optimal solution is an integer and the corresponding objective function value is larger than problem *D*'s, this solution is the optimal solution. If the optimal solution is not integer, but the objective function value is larger than *D*, then continue to branch this problem. While if it is smaller than *D*, then branch the problem *D*.
We repeat the third to fifth steps above until the optimal solution is found.
In order to make the procedure above more clear ,we give an example to explain in detail next.

Example 2
A truck with a maximum capacity of 10 tons is used to carry three kinds of goods (the quantity of each kind of goods can be loaded more than one). As for each goods, the weight of

per unit and its value are shown in Table 4. Then which goods should be put to maximize the total value in the knapsack?

Table 4: The weight and value of three goods

Number of the goods(*i*)	1	2	3
Weight of per unit /t	3	4	5
Value of per unit(c_i)	4	5	6

Solution:

According to the description of the problem above, we establish an integer linear programming model: Supposing that x_i represents the quantity of the *i* th kind of goods loaded in the truck, and *z* represents the total value of the goods:

$$\max z = 4x_1 + 5x_2 + 6x_3$$
$$s.t.\begin{cases} 3x_1 + 4x_2 + 5x_3 \le 10 \\ x_i \ge 0 \text{ and } x_i \text{ is an integer} \end{cases} \quad (i = 1,2,\cdots,n) \qquad (7)$$

We enter the code in MATLAB to implement the branch and bound method, and the results obtained are as follows:

```
x =
 2.0000
 1.0000
 0
fval =
 13.0000
0.697441 seconds have passed.
```

Therefore, the optimal solution calculated is $x_1 = 2, x_2 = 1, x_3 = 0$, and the maximum value of the objective function is 13, which means that when the truck loads the 1th and 2th kind of goods with a weight of 2t and 1t respectively, the value reaches the maximum of 13.

3. The dynamic programming method to solve knapsack problem

Since the knapsack problem has multiple stages to put different types of items, and its purpose is to maximize the value of all items put in the knapsack, it is necessary to make the best decision from a global perspective. Therefore, the knapsack problem is also a multi-stage decision problem, which can be solved by dynamic programming method.

There are two basic algorithm in solving dynamic programming problems: forward dynamic programming algorithm and backward dynamic programming algorithm. Next we will study their basic principles and their application in the knapsack problem.

3.1 The forward dynamic programming algorithm for knapsack problem

Supposing there are *n* items, then the forward dynamic programming algorithm can be described as follows:

- Stage *k*: According to the order of loading we divide the whole procedure into *n* stages. We can put only one kind of the items in the knapsack at each stage, therefore the value of k is $1,2\cdots,n$;
- State variable s_{k+1}: At the beginning of the stage *k*, the total weight of the previous *k* items allowed in the backpack is s_{k+1};
- Decision variable x_k : the quantity of *k* th items loaded in the knapsack;
- State transfer equation: $s_k = s_{k+1} - a_k x_k$ (8)
- The decision sets:
  $D(s_{k+1}) = \{x_k \mid 0 \le x_k \le [s_{k+1}/a_k], x_k \text{is an integer}\}$ (9)
- The optimal index function $f_k(s_{k+1})$ represents the maximum value of previous *k* kinds items with the constriction that the weight of the k+1 kind of item is not more than $s_{k+1}kg$, and the optimal strategy is used to load only the first k items. The recurrence equation is as follows:

$$\begin{cases} f_k(s_{k+1}) = \max\limits_{x_k=0,1,\cdots,[s_{k+1}/a_k]} \{c_k(x_k) + f_{k-1}(s_{k+1} - a_k x_k)\} \quad (k = 0,1,\cdots,n) \\ f_0(s_1) = 0 \end{cases}$$
(10)

We perform the algorithm from $k = 1$ to $k = n$ to obtain the maximum value of the problem, and then the optimal solution can be deduced inversely.

Next we apply the forward dynamic programming algorithm to solve example 2.

Solution:

There are 3 stages in this dynamic planning, and the 1tth, 2th and 3th kinds of goods are put in successively; the decision variable x_1, x_2, x_3 are the quantities of the 1th, 2th and 3th kinds of goods; weight of per unit $a_1 = 3, a_2 = 4, a_3 = 5$ and the value of per unit $c_1 = 4, c_2 = 5, c_3 = 6$ are substituted into (8) (9) (10) then the transfer equations are obtained as follows:

$$k = 1,\ f_1(s_2) = \max_{0 \le 3x_1 \le s_2} \{4x_1\} \qquad (11)$$

$$k = 2,\ f_2(s_3) = \max_{0 \le x_2 \le s_3/4} \{5x_2 + f_1(s_3 - 4x_2)\} \qquad (12)$$

$$k = 3,\ f_3(10) = \max_{x_3=0,1,3} \{6x_3 + f_2(10 - 5x_3)\} \qquad (13)$$

according to the steps above, we write the MATLAB code in to implement the forward dynamic programming algorithm, and the results obtained are as follows:

```
step1 =
 0 1 2 3 4 5 6 7 8 9 10
 0 0 0 4 4 4 8 8 8 12 12
 0 0 0 1 1 1 2 2 2 3 3
step2 =
 0 1 2 3 4 5 6 7 8 9 10
 0 0 0 4 5 5 8 9 10 12 13
 0 0 0 0 1 1 0 1 2 0 1
step3 =
 0 1 2 3 4 5 6 7 8 9 10
 13 13 13 13 13 10 10 10 10 10 12
 0 0 0 0 0 1 1 1 1 1 1
```

0.107600 seconds have passed.
It can be seen from the second line of "step 3" that the maximum value of the original problem is 13. In this case, the weight of the 3th kind of the goods is 0t, and the 1th and 2th kind of the goods can be calculated as 2t and 1t.

3.2 The backward dynamic programming algorithm for knapsack problem

Supposing there are *n* items, then the backward dynamic programming algorithm can be described as follows:

- Stage *k*: According to the order of loading we divide the whole procedure into *n* stages. We can put only one kind of the items in the knapsack at each stage, therefore the value of k is $1,2\cdots,n$;
- State variable s_{k+1}: At the beginning of stage *k*, the maximum weight allowed in the knapsack in the next stages is s_{k+1};
- Decision variable x_k : the quantity of *k* th items loaded in the knapsack;
- State transfer equation: $s_{k+1} = s_k - a_k x_k$ (14)
- The decision sets: $D(s_k) = \{x_k \mid 0 \le x_k \le [s_k / a_k], x_k \text{is an integer}\}$ (15)
- The optimal index function $f_k(s_k)$ represents the maximum value of the items allowed to be loaded in the knapsack from the stage *k* to the stage *n* when the initial state variable is s_k. The transfer state equation is as follows:

$$\begin{cases} f_k(s_k) = \max\limits_{x_k=0,1\cdots,[s_k/a_k]} \{c_k(x_k) + f_{k+1}(s_k - a_k x_k)\} & (k = 1,2\cdots,n) \\ f_{n+1}(s_{n+1}) = 0 \end{cases}$$

(16)

The following we use backward dynamic programming algorithm to solve example 2.
Solution:
There are 3 stages in this dynamic planning, and the 1tth, 2th and 3th kinds of goods are put in successively; the decision variable x_1, x_2, x_3 are the quantities of the 1th, 2th and 3th kinds of goods; weight of per unit $a_1 = 3, a_2 = 4, a_3 = 5$ and the value of per unit $c_1 = 4, c_2 = 5, c_3 = 6$ are substituted into (14) (15) (16) then the transfer equations are obtained as follows:

$$k = 3,\ f_3(s_3) = \max_{0 \le 5x_3 \le s_2} \{6x_3\} \quad (17)$$

$$k = 2,\ f_2(s_2) = \max_{0 \le x_2 \le s_2/4} \{5x_2 + f_3(s_2 - 4x_2)\} \quad (18)$$

$$k = 1,\ f_3(10) = \max_{x_3=0,1,3} \{4x_1 + f_2(s_1 - 3x_1)\} \quad (19)$$

We use MATLAB realize the above process more quickly. And the results obtained are as follows:
optimal solution:13
put 2 item1 in the bag
put 1 item2 in the bag
put 0 item3 in the bag
0.071611 seconds have passed.
From the above results, it can be seen that the maximum value of the items loaded in the truck is 13 when the weight of the 1th, 2th, and 3th are 2t, 1t, 0t, which is the same as the result of the forward dynamic programming algorithm. However, the MATLAB program of the backward dynamic programming algorithm takes about 0.072 seconds, which is shorter than the 0.1 second time used by forward dynamic programming, indicating that process of the backward dynamic programming algorithm is easier for this problem.

4. The analysis and comparison of integer programming method and dynamic programming method

Though the results above, it can be concluded that both integer programming method and dynamic programming method can find the optimal solution and the maximum value of the objective function for the knapsack problem, but they have their own characteristics.
When using the integer programming method to solve the knapsack problem, it is necessary to determine whether the knapsack problem is a 0-1 integer programming problem or a pure integer programming problem. When it is a 0-1 integer programming problem, it is suitable to use implicit enumeration method to solve. As for implicit enumeration method, it can be seen from the solution process of example 1 that we should list all the situations of the knapsack problem first, then calculate the objective function values of them in turn, judge whether they conform the constraint conditions so that to determine whether the value of the objective function can be a lower bound, causing a new constraint condition for the next situation. After repeating the above steps, the optimal solution is the solution with the maximum of lower bound of the objective function. When the knapsack problem is a pure integer programming problem, it is more appropriate to use the branch and bound method to solve it, that is, first we should branch an independent variable, generating two subsequent problems, and compare the values of the objective function of the subsequent problems. Repeat the process until the solutions obtained are all integers and the objective function value is the largest.
It can be seen from the solution process of example 2 that the built-in functions in MATLAB can be used directly when using the branch and bound method to solve the problem, and the running time is about 0.697 seconds.
When using the dynamic programming method to solve the knapsack problem, firstly it is necessary to clarify the total stages, and then define the state variables, decision variables, state transition equations, allowable decision sets and index functions. Then we should choose the forward dynamic programming algorithm or the backward dynamic programming algorithm, whose state variables, decision variables, state transition equations, allowable decision sets and index functions are all different, which need to be defined according the practical problem. Additionally, the calculation direction of the forward dynamic programming is from the first stage to the last stage, but the direction of decision-making is from back to forward. While the calculation direction of the backward dynamic programming is from the last stage to the first stage, but the direction of decision-making is from forward to backward

In order to implement the forward dynamic programming algorithm or the backward dynamic programming algorithm, we can use MATLAB software. Because there is no built-in function in MATLAB for this two algorithms, long programs needs to be written. The running time of the two algorithms is 0.1 seconds and 0.072 seconds respectively.

5. Conclusions

Knapsack problem is an important problem in operations research, which is closely related to many practical problems. There are many methods to solve the knapsack problem, but the most commonly used methods are integer programming method and dynamic programming method. This article has compared and analyzed the two methods with the combination of practical problems, then the conclusions can be obtained as follows:

(1) When use the integer programming method to solve the knapsack problem, we should first determine whether the problem is a 0-1 integer programming problem or a pure integer programming problem. When it is a 0-1 programming problem, it is easier use the implicit enumeration method which has few steps and can be calculated manually. However, when the knapsack problem has many stages and is complicated, this method is not applicable. When the knapsack problem is a pure integer programming problem, the commonly used method is branch and bound method.

(2) The MATLAB program of the integer programming method is simple and the running time is short. While the MATLAB program of the dynamic programming method is more complicated and the running time is relatively long.

(3) The dynamic programming method requires us to divide the stages first according to the problem;

(4) The dynamic programming method has two algorithms: forward dynamic programming algorithm and backward dynamic programming algorithm, whose the meanings and values of variables are different. And the complexity for solving one knapsack problem using two algorithms may be different, which should be consider according to the practical problem in advance.

(5) When using the dynamic programming method to solve the knapsack problem, forward dynamic programming algorithm can be used to find the optimal solutions from the initial stage to any middle stage, and the backward dynamic programming algorithm can be used to find the optimal solutions from any middle stage to the last one.

In conclusion, the integer programming model is more suitable for the knapsack problem which only needs to calculate the final result of the loading items, while the dynamic programming method is more suitable for the one which pays attention to the middle optimal solutions.

6. References

1. Merkle R, Hellman M. Hiding Information and Signatures in Trapdoor Knapsacks. IEEE Transactions on Information Theory. 1978; 24(5):525-530.
2. Fernando AM, Jairo AM. Analysis of Divide-and-Conquer strategies for the 0–1 minimization knapsack problem. Springer US. 2020, 40.
3. Thomas S, Sebastian MB. Corrigendum to “Empirical Orthogonal Constraint Generation for Multidimensional 0/1 Knapsack Problems”. Elsevier B.V. 2020; 286:2.
4. Huynh DQ, Nguyen CT, Tran HN. The Continuous Knapsack Problem with Capacities. Operations Research Society of China, 2020.
5. Djeumou FF, Kaparis K, Letchford AN. A cut-and-branch algorithm for the Quadratic Knapsack Problem. Elsevier, 2020.
6. Huihong W, Shuqu Q, Yanmin L, Dong W, Benhua G. An immune-based response particle swarm optimizer for knapsack problems in dynamic environments. Springer Berlin Heidelberg, 2020.
7. Thomas S, Sebastian MB.. Empirical orthogonal constraint generation for Multidimensional 0/1 Knapsack Problems. Elsevier B.V. 2020; 282:1.
8. Zhengtian W, Baoping J, Hamid RK.. A logarithmic descent direction algorithm for the quadratic knapsack problem. Elsevier Inc. 2020; 369.
9. Nicolás A, Carlos R, Carlos C, Victor P. Automatic design of specialized algorithms for the binary knapsack problem. Elsevier Ltd. 2020; 141.
10. Inarejos O, Hoto R, Maculan N. An integer linear optimization model to the compartmentalized knapsack problem. International Transactions in Operational Research. 2019; 26(5).
11. John JQ. and Valério G, Robinson SV. A strong integer linear optimization model to the compartmentalized knapsack problem. International Transactions in Operational Research. 2019; 26(5).
12. Fenlan W. A new exact algorithm for concave knapsack problems with integer variables. Taylor & Francis. 2019; 96(1).
13. Qian SQ, Wu HH & Lin Y. Clonal repair immune algorithm for solving high-dimensional dynamic knapsack problem. Computer Engineering. 2017; 43(09):220-227.
14. Lan WF, Wu ZY & Yang B. An improved dynamic programming algorithm for the knapsack problem. Journal of South-Central University for Nationalities. 2016; 35(04):101-105.
15. Wu HH, Qian SQ & Xu GF. A patched binary differential evolution algorithm for solving high-dimensional dynamic 0-1 knapsack problem. Application Research of Computers. 2016; 33(10):2941-2945.
16. Andrew M. and Patrick J.. Average-Case Performance of Rollout Algorithms for Knapsack Problems. Springer US. 2015; 165(3).
17. Li JM, Fu YF. Analysis of improved dynamic programming algorithm for solving 0-1 knapsack problem. Journal of Northwest University. 2014; 44(05):729-732.
18. José RF, Gabriel T, Margaret MW. Labeling algorithms for multiple objective integer knapsack problems. Elsevier Ltd. 2009; 37(4).
19. Kamlesh M, Prahalad V. A new lower bound for the linear knapsack problem with general integer variables. Elsevier B.V.. 2006; 178(3).