ISSN: 0067-2904

Improved Weighted 0-1 Knapsack Method (WKM) to Optimize Resource Allocation

Maha A. Hammood Alrawi* , Israa Tahseen Ali, Olaa Amer Saied
Department of Computer Science, University of Technology, Baghdad, Iraq.

Abstract

In this paper an improved weighted 0-1 knapsack method (WKM) is proposed to optimize the resource allocation process when the sum of items' weight exceeds the knapsack total capacity .The improved method depends on a modified weight for each item to ensure the allocation of the required resources for all the involved items. The results of the improved WKM are compared to the traditional 0-1 Knapsack Problem (KP). The proposed method dominates on the other one in term of the total optimal solution value of the knapsack .

Keywords: knapsack problem, dynamic programming ,optimal solution,resource allocation.

طريقة الحقيبة الموزونة المحسنة 0−1 *(WKM)* لأمثلية تخصيص الموارد

مها عبد الكريم حمود الراوي* ، إسراء تحسين علي ، علا عامر سعيد
قسم علوم الحاسوب، الجامعة التكنولوجية ، بغداد، العراق.

الخلاصة

اقترحت هذه الورقة طريقة الحقيبة الموزونة المحسنة (0−1) لامثلية تخصيص الموارد في الحالات التي يتجاوز فيها مجموع اوزان المفردات التي تتضمنها الحقيبة لاجمالي قيمة الحقيبة . تعتمد هذه الطريقة المحسنة على تعديل الوزن لكل مفردة لضمان تخصيص الموارد اللازمة لجميع المفردات المشمولة ضمن المشكلة. وقد تم مقارنة نتائج تطبيق الطريقة المقترحة مع مشكلة الحقيبة التقليدية 0−1، واوضحت النتائج هيمنة الطريقة المقترحة على الطريقة الاخرى من حيث إجمالي قيمة الحل الامثل للحقيبة.

Introduction

Today's world is characterized by technical, economic, social developments, big size of the installations, and ever changing business environment which led to the complexity and uncertainty**.** Rapid developments in various fields made an urgent need for adopting scientific and logical tools which help in decision making and solving problems that are faced by business organizations**.**

One of the Dynamic Programming (DP) algorithms is Knapsack Problem method (KP), which enables decision-makers to derive optimal solutions that effectively contribute to the decision-making process. In this paper we suggested an improvement on traditional knapsack algorithm to concern with multi factors and resources that ensure optimal decision.

A problem frequently encountered is how to allocate limited resources to competing, necessary, and profitable items. These resources are frequently raw materials, workers, manufacturing times, or money that are needed to produce products. The optimal resource allocation requires specifying the total amount of the resources and their weights to be distributed for all the required items [1].

*Email: maha_alrawi@yahoo.com

The Knapsack problem(or rucksack problem) is a general resource allocation model in which a single resource is assigned to a number of alternatives with the objective of maximizing the total return. The model is one of the most important classes of applications of DP. The model is a distribution of effort problem that has a linear objective function and a single linear constraint [2].

Suppose you have a bag that consists of several items (i= 1,2,3,...,n) carried in bag ,each item has a weight (W_i) and profit (P_i) obtained by putting items in bag , all P_i and W_i are integer positive numbers. However the sum of weights for all items is less than the capacity (C)of the bag. This model determines the most valuable items to be carried in the bag and their quantities, weights, and profits [3].The KP model has been applied to many real life applications either a standalone model or as a combination of models[4].

Related work

The proposed method is concerned with specific topics of resource allocation that have been studied in related literatures.

In1996, **Yu Gang** [5], studied the Max-Min Knapsack (MNK) problem as a NP-hard for an unbounded number of scenarios and pseudo polynomial solvable for a bounded number of scenarios. The effective of lower and upper bounds were generated by surrogate relaxation. The ratio of these two bounds is shown to be bounded by a constant for situations where the data range is limited to be within a fixed percentage from its mean. A branch-and-bound algorithm has been implemented to efficiently solve the MNK problem to optimality.

In 2009, Campegiani and Presti [6], suggested a generalization model of the classical 0/1 Knapsack Problem. They developed a heuristic to obtain very near optimum solutions in a timely manner.

In 2013,Zhang et al. [7], proposed a new bio-inspired model to solve problems. The proposed method has three main steps. First, the 0-1 knapsack problem is converted into a directed graph by the network converting algorithm. Then, for the purpose of using the amoeboid organism model, the longest path problem is transformed into the shortest path problem. Finally, the shortest path problem can be well handled by the amoeboid organism algorithm. Numerical examples are giving to illustrate the efficiency of the proposed mode.

In 2015, Rooderkerk and Heerde[8],developed a robust approach to optimize retail assortments since retailers face the difficult task of designing a portfolio of products that balances risk and return. They proposed a novel, efficient and real-time heuristic depends on 0-1 Knapsack that solves the problem and offers an optimal balance. The heuristic constructs an approximation of the risk-return Efficient Frontier of assortments

Problem Statement

The 0-1 knapsack problem is a combinatorial optimization problem . It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items .Given a set of items, each item has a weight (w_i) and a profit (p_i), The problem determines the quantity of each item to be included in a knapsack so that the total weight is less than or equal to a given limit, and the total capacity value(C)of the problem could be as large as possible. Ander these assumptions, the general model of KP is formulated as in the following equations [9]:

Maximize $\sum_{i=1}^{n} p_i x_i$ .....(1)

Subject to: $\sum_{i=1}^{n} w_i x_i \leq c$ .....(2)

where

x_i: $\begin{cases} 1 & \text{if } \textit{the item i is selected} \\ 0 & \textit{if not.} \end{cases}$

w_i: the unite weight for item i= 1, 2,3. . .,n
p_i: the unite profit for item i= 1, 2,3 . . . n
C: the total capacity of Knapsack.

One-dimensional 0-1 KP (S,C) could be explained as follows : suppose there is combination S of a 0-1 KP and n various items (i= 1, 2,3. . .,n) . Item (i) obtained a profit p_i , weight w_i , and a capacity C. Where p_i , w_i and C are integer positive numbers, and ($w_n \leq C$),where($n \geq i \geq 1$) .

Let $f_n(c)$ in equation (3) be the optimal solution for the problem [10].

$f_n(c)=\max\{f_{n-1}^*(C), f_{n-1}^*(C - w_n)+p_n\}$ ..... (3)

This equation can be generalized as follows

$$f_0(X) = \begin{cases} 0 & x \geq 0 \\ -\infty & x < 0 \end{cases} \text{.....(4)}$$

$$f_i(x) = max\{f_{i-1}(x), f_{i-1}(x - w_i) + p_i\}, \quad for\ all\ x \text{..... (5)}$$

Where i=1,2,3.....,n

A Single-processor Machine for the 0-1 knapsack problem in Dynamic Programming Method

Dynamic Programming (DP) solves the problem by producing "f_1, f_2, …,f_n" sequentially. As mentioned in the previous section, $f_i(x)$is a monotone no lessening basic step work. "$f_i(x)$" may be exemplified as the set SP_i From claiming rows from the coordination of that phase focuses of "$f_i(x)$". The size of the set *Si*,(i.e. $|SP_i|$) , is not greater than "C+1" and rows should be planned in growing arrangement x while $f_i(x)$. The series of sets, "SP_0,SP_1,...,SP_n", is a history of the DP and that should be backtracked through "Algorithm 2" to get the solution vector **x**. "Algorithm 1"result sets "SP_1,SP_2, …,$SP_{n''}$ as follows:

Algorithm 1: forward part of dynamic programming
Input:W_{item} : Weight of each item ,P_{item}: Profit of each item, C : Capacity **Output**: SP : array of subproblem to find an optimal solution,
Step1: $S_0 \leftarrow \{(0,0)\}$ ***Step2:*** $for\ item = 1\ to\ n\ do$ $SP_{\text{item}} \leftarrow \{(P + P_{\text{item}}, W + W_{\text{item}}) \mid (P, W) \in SP_{\text{item}-1}, W + W_{\text{item}} \leq C\}$ $SP_{\text{item}} \leftarrow merge(SP_{\text{item}-1}, SP_{\text{item}})$ *end for*

Here ,(P_i) is the value of the profit function "$f_{i-1}(x)$" while a weight "$x=C$ ". The *merge* procedure merges two lists (SP_{i-1} and *SPi)*to create list *Si*. During the merging, if $SPi' \cup SP_{i-1}$ contains two tuples (P_j,W_j) and (P_k,W_k) such that $P_j \leq P_k$ and $W_j \geq W_k$, i.e. (P_k,W_k) dominates (P_j,W_j), followed by (P_j,W_j) is not needed.

The *P* is the optimal solution , if (P,W) is the end row the solution vector **x** such using equation(1& 2) is set by searching out of the sets SP_n,SP_{n-1}, . . . ,This task is performed by Algorithm 2[10].

Algorithm2:backtracking part of dynamic programming
Input: SP : array of subproblem **Output**: X : 0 or 1 select item or not

Step1: *(P,W)←end row in SP*
 	
Step2: **for** item □ *n* **down to** 1 **do**
if (P-P_{item},*W-*W_{item}*)* ∈ $SP_{\text{item}-1}$*then*
xi←1, W←W- W_{item}*, P←P-*P_{item}
else
x←0
endif
end for

For the 0–1 Knapsack problem, when we have a single instance of each item, we cannot use the same relation, while we do not know if item *i* is already used in the optimal solution SP(P,W). To keep track of which items are used, we not only look at smaller knapsacks, but also at fewer items. Algorithm (3) describes this [11].

Algorithm 3- traditional 0-1 Knapsack Algorithm

Input:W_{item} : Weight of each item ,P_{item}: Profit of each item, C : Capacity
Output: SP : array of subproblem to find an optimal solution, X : 0 or 1 select item or not

Step1: $SP_{i,j} \leftarrow \{(0,0)\}$ // item : number of item , TN :total number of item
Step2: for item = 1 to TN do //Capcount: Capacity count
for Capcount ← 0 to C do
  if $W_{item} \leq C$ then //SP : subproblem
    if $P_{item} + SPD[item-1, Capcount - W_{item}] > SP[item-1, Capcount]$then
      $SPD[item, Capcount] \leftarrow P_{item} + SP[i-1, j-W_{item}]$
else
      $SP[item, Capcount] \leftarrow SP[item-1, Capcount]$
Endif
else
  $SP[item, Capcount] \leftarrow SP[item-1, Capcount]$ // W_{item}>C
Endif
end for // Capcount
end for // item
Step3: item←TN,Capcount←C
Step4: while item and Capcount> 0
  if SP[item–1,Capcount] ≠ SP[item,Capcount] then
    item← item–1, Capcount←Capcount-W_{item}
X_{item}=1
  else
    item←item–1
X_{item}=0

endif
endwhile

Improved Weighted 0-1 Knapsack Method (WKM)

Our proposed method provides an optimal solution for resources allocation as a DP problem when the total items weights exceed the total capacity. First , a fraction of capacity to total original weights (Wf) is calculated as in equation (6). The improved weight that is proposed in WKM (NW_i) is a ratio of original weight as in equation (7) . The benefit of the new proposed weight (NW_i) ensures allocating the available resources for all items.

$$Wf = \frac{C}{\sum_{i=1}^{n} wi} \quad \text{.....(6)}$$

$$NW_i = Wf \times wi \quad \text{.....(7)}$$

By applying these equations (6 and 7) to algorithm 4, the value of NW_i will be the optimal weight to ensure allocating the required resources for all the involved items.

Algorithm 4- Improved 0-1 Knapsack Algorithm

Input:W_{item} : Weight of each item ,P_{item}: Profit of each item, C : Capacity
Output: W : new weight before update , SP : array of subproblem to find an optimal solution, X : 0 or 1 select item or not

Step1: $SP_{i,j} \leftarrow \{(0,0)\}$ // item : number of item , TN :total number of item
Step2:for item = 1 toTN − 1 do
sum ←sum + W_{item}
end for
fraction ← C / sum
for item = 1 toTN − 1 do
$W_{item} \leftarrow W_{item}$ * fraction
End for
Step3: for item = 1 toTN do //Capcount: Capacity count
for Capcount ← 0 to C do
if $W_{item} \leq C$ then //
if $P_{item} + SP[item - 1, Capcount - W_{item}] > SP[item - 1, Capcount]$then
$SP[item, Capcount] \leftarrow P_{item} + SP[i - 1, j - W_{item}]$
else
$SP[item, Capcount] \leftarrow SP[item - 1, Capcount]$
Endif
else
$SP[item, Capcount] \leftarrow SP[item - 1, Capcount]$ // W_{item}>C
Endif
end for // Capcount
end for // item
Step4: item←TN,Capcount←C
Step5: while item and Capcount> 0
if SP[item−1,Capcount] ≠ SP[item,Capcount] then
item← item−1, Capcount←Capcount-W_{item}
X_{item}=1
else
item←item−1
X_{item}=0

endif
endwhile

Experimental Results and Discussions

In this section, the results obtained by implementing the improved algorithm WKM will be discussed in details to explain the role of the new proposed weight (NW_i) clearly.

Consider the problem (X) with the following given data in Table-1, where n is number of items (n=4) and C is the total capacity (C=8). Each item (i) has a knapsack weight (W_i) , and a knapsack profit (P_i) obtained by allocating required resource to the specified item i. All P_i 's and W_i 's are positive integer numbers. Then, the subproblem SP[TN ,Capcount] in algorithm 4 will be computed to find optimal solution for the list (SP) of the n items

Table 1- problem (X)

Item	profit	Weight
1	15	1
2	10	5
3	9	3
4	5	4
Capacity=8		

After implementing algorithm (3)(traditional 0-1 Knapsack Algorithm)using the considered problem (X) in Table-1, set of results are obtained experimentally as displayed in Table-2

Table -2-Experimental results for forward computations of algorithm (3)

Item	**Capacity remaining**								
	K=0	**K=1**	**K=2**	**K=3**	**K=4**	**K=5**	**K=6**	**K=7**	**K=8**
Initial	0	0	0	0	0	0	0	0	0
i=1	0	15	15	15	15	15	15	15	15
i=2	0	15	15	15	15	15	25	25	25
i=3	0	15	15	15	24	24	25	25	25
i=4	0	15	15	15	24	24	25	25	29

The step by step backtracking computations of algorithm (3) is shown as bellow :
K=8
i=4 →SP(4,8) ≠ SP(3,8) → x = 1
K= K - w → 8-4 = 4
i=3→ SP(3,4) ≠ SP(2,4)→ x = 1
K= K - w → 4-3 = 1
i=2→ SP(2,1) = SP(1,1) → x = 0
K= K - 0 → 1
i=1 →SP(1,1) ≠ SP(0,1)→ x = 1
K= K - w → 1-1 = 0
x = (1,0,1,1) ,select item 1,3,4 , the optimal solution value 29.

On the other hand, when the Improved knapsack (WKM) algorithm (4) is implemented using problem (X) that shown in Table-1, the fractional weights (Wf) will be calculated as in equations (6) and (7) . So , the new improved weights (NW_i) are shown in Table-3
To show the mechanism of applying equations (6) and (7) , consider items 1 and 2 .
sum of weights =13
Wf= 8/13= 0.6 ..... as in equations (6)
NW_1 = 1*0.6 = 0.6 round = 1 ..... as in equations (7)
NW_2= 5*0.6 = 3 as shown in Table-3

Table 3- problem (X) with NW_i

Item	**profit**	**Wight**	**New Weight**
1	**15**	**1**	**1**
2	**10**	**5**	**3**
3	**9**	**3**	**2**
4	**5**	**4**	**2**
Capacity=8			

After the improved calculated knapsack the new weights as shown in Table-3 , the optimal solution value as shown in Table -4 is improved equal to 39 but before implementing knapsack the optimal solution value as shown in Table -2 was 29 . This is illustrated, when this clarify that the update on the weights values lead to a better solution, in addition to the benefit from the largest number of items.

Table -4-Experimental results of forward computations of algorithm (4)									
Item	**Capacity remaining**								
	K=0	**K=1**	**K=2**	**K=3**	**K=4**	**K=5**	**K=6**	**K=7**	**K=8**
I=0	0	0	0	0	0	0	0	0	0
i=1	0	15	15	15	15	15	15	15	15
i=2	0	15	15	15	25	25	25	25	25
i=3	0	15	15	24	25	25	34	34	34
i=4	0	15	15	24	25	29	34	34	39

The backtracking computations of algorithm (4)
K=8
i=4 →SP(4,8) ≠ SP(3,8) → x = 1
K= K - w → 8-2 = 6
i=3→ SP(3,4) ≠ SP(2,4)→ x = 1
K= K - w → 6-2 = 4
i=2→SP(2,1) ≠ SP(1,1) → x = 1
K= K - w → 4-3 = 1
i=1 →SP(1,1) ≠ SP(0,1)→ x = 1
K= K - w → 1-1 = 0
x = (1,1,1,1) ,select item 1,2,3,4 , the optimal solution value 39.

Depending on modifying each item weight through equation (3) , this led to Increase the value optimal solution of SP(P,W) from Table -2 equal 29 by algorithm (3) from Table -4 equal 39 by algorithm (4). The new optimal solution allows to allocate the required resources to more items included in the KP .

Conclusions and future works

In this paper, a modified 0-1 Knapsack method called Weighted 0-1 Knapsack Method (WKM) is proposed. The proposed method provides an optimal solution for resources allocation as compared with traditional 0-1 KP , when the sum of items weights exceeds the total capacity. A fraction of capacity to total original weights . The improved weight that proposed in WKM is a ratio of original weight The benefit of the new proposed weight ensures allocating the available resources for all items. The proposed new weight ensures allocating resources for all items involved in dynamic programming problem using 0-1 Knapsack method.

As a future work, we could gain more optimality by using multi-objective knapsack or online knapsack problems to solve resource allocation problems .Network distribution systems can be used too to implement more than one task at a time which yields to decrease the time spent in resource allocation .

References

1. Sanford, M. Roberts .**1964**. *Dynamic Programming in Chemical Engineering and Process Control*, Publisher: Academic Press Inc., U.S, 1st Edition.
2. Taha, H.A. **2008**. *Operations Research: An introduction*. India: Prentice-Hall publishers ,8th Edition.
3. Smith, D.K. **1991**. *Dynamic Programming: A practical Introduction*, London, England: Ellis Horwood publishers.
4. Owoloko, E. A., Sagoe E.T. **2010.** Optimal advert placement slot – using the knapsack problem model. *American Journal of Scientific and Industrial Research*, **1**(1): 51-55 .
5. Yu, Gang .**1996**. On the Max-Min 0-1 Knapsack Problem with Robust Optimization Applications , *Journal Operations Research*, **44**(2): 407–415.
6. Paolo, Campegiani, Francesco, Lo Presti .**2009.** *A General Model for Virtual Machines Resources Allocation in Multi-tier Distributed Systems*, ICAS '09. Fifth International Conference on Autonomic and Autonomous Systems, vol. 00, no. undefined, pp: 162-167.

7. Zhang, X., Huang, Sh., Hub, Y., Zhang Y., Mahadevan, S. and Deng, Y. **2013**, Solving 0-1 knapsack problems based on amoeboid organism algorithm. *Applied Mathematics and Computation*, **219**(19): 9959–9970.
8. Robert, R. P. and Van Heerde, H.J. **2016.** Robust Optimization of the 0-1 Knapsack Problem: Balancing Risk and Return in Assortment Optimization. *European Journal of Operational Research*, **250**(3): 842-854 .
9. Jong, Lee, Eugene, Shragowitz and Sartaj, Sahni .**1988**. A hypercube algorithm for the 0/1 knapsack problem. *Journal of Parallel and Distributed Computing*, **5**(4): 438-456.
10. Jelke, J. and van Hoorn. **2016.** Dynamic Programming for Routing and Scheduling: optimizing sequences of decisions. Ph.D. thesis, VU University, Amsterdam.