International Journal of Informatics and Communication Technology (IJ-ICT)
Vol.2, No.1, January 2013, pp. 9~16
ISSN: 2252-8776

Optimal Solution Of Minmax 0/1 Knapsack Problem Using Dynamic Programming

Ani Dijah Rahajoe*, Edi Winarko**
* Departement of Informatics Engineering, Bhayangkara Surabaya University
** Faculty Of Mathematics and Natural Sciences, Universitas Gadjah Mada Yogyakarta

Article Info

Article history:

Received Aug 16th, 2012
Revised Sept 30th, 2012
Accepted Oct 15th, 2012

Keyword:

Integer knapsack
Minmax
Dynamic programming

ABSTRACT

Knapsack problem is a problem that occurs when looking for optimal selection of objects that will be put into a container with limited space and capacity. On the issue of loading goods into the container, optimal selection of objects or items to be sent must fulfilled to minimize the total weight of the capacity or volume limits without exceeding the maximum capacity of containers that have been determined. The types of knapsack that has been discussed so far is only to maximize the use not to exceed the limits specified capacity so it cannot be applied to the problem. This study aims to develop a dynamic programming algorithm to solve the MinMax 0/1 knapsack, which is an extension of the *0/1* knapsack with minimal and maximal constraints. The result of study showed that the solution of the MinMax *0/1* knapsack problem using dynamic programming can be used to generate the optimal solution to the problem of loading goods into the container such that the minimum and maximum capacity constraints are met.

Corresponding Author:

Edi Winarko,
Faculty Of Mathematics and Natural Sciences, Universitas Gadjah Mada Yogyakarta,
Bulaksumur, Yogyakarta 55281, Indonesia.
Email: ewinarko@ugm.ac.id

1. INTRODUCTION

Optimization is one of the mathematical disciplines that aim to find the maximum or minimum value of a function with certain limitations (constraints). One example is optimization knapsack problem that defined as follows: If there's several item, each with their own weight and value, the knapsack problem is to determine the item that will be incorporated into the knapsack such that the total weight is less than or equal to the limit and have maximum total value. There are two categories known in knapsack problem, the fractional knapsack and integer knapsack. In the fractional knapsack objects can be inserted into the container in fraction, while the integer knapsack items should be included as a whole. One form of the integer knapsack that is often discussed is the 0/1 knapsack [1], which is mathematically formulated in equation (1).

$$\text{Maximize} \quad \sum_{j=1}^{n} p_j x_j \qquad (1)$$
$$\text{Requirement} \quad \sum_{j=1}^{n} w_j x_j \le M$$
$$x_j \in \{0,1\}, \; j=1,..,n.$$

In this paper, we propose an extension of the *0/1* knapsack problem, called MinMax *0/1* knapsack problem, which is formulated in equation (2).

$$\text{Minimize} \quad \sum_{j=1}^{n} p_j x_j \qquad (2)$$
$$\text{Requirement} \quad M_1 \le \sum_{j=1}^{n} w_j x_j \le M_2$$
$$x_j \in \{0,1\}, j=1,..,n$$

where M_1 is the minimum limit of the knapsack and M_2 is the maximum limit.

One example of the MinMax *0/1* knapsack problem is the loading of goods into a container which aims to minimize the use of container space available on the delivery of goods between islands or countries. In this problem the items are inserted into the container must meet the minimum limit and not exceed the maximum capacity.

2. RELATED WORKS

The general knapsack problem is a problem of selecting objects or items (each of which has a weight and value or profit) to be put in a knapsack so that the total profit is maximum but the total weight does not exceed the maximum capacity of knapsack. There several types of knapsack problems, such as bounded knapsack, unbounded knapsack, and integer knapsack problem, and fractional knapsack problem [2]. Integer knapsack has several types including multidimensional knapsack problems, two-dimensional knapsack problem, precedence constraint knapsack problem [3], disjunctively constraint knapsack problem [4], multiple choice knapsack problem, the knapsack sharing problem [5], the quadratic knapsack problem [6], and Max-min knapsack problem [7] [8].

The 0/1 knapsack problem is a special case of integer knapsack problem, in which items should be put in the knapsack as a whole. Research on the 0/1 knapsack problem solving has been widely applied, such as by using brute force algorithms, greedy, branch and bound, and dynamic programming [9] [10]. Dynamic programming has been used widely to solve extensions of 0/1 knapsack problem. As examples, a dynamic programming algorithm is used to solve a bilevel knapsack problem, in which the leader determines the knapsack's capacity in order to maximize his profit, while the follower faces a 0/1 knapsack problem involving the capacity set by the leader [11]. The dynamic programming is also used to solve the knapsack sharing problem [12]. Recently, dynamic programming based algorithms is proposed for solving the discounted 0/1 knapsack problem [13]. The discounted 0/1 knapsack problem (DKP) is an extension of the classical 0/1 knapsack problem that consists of selecting a set of item groups where each group includes three items and at most one of the three items can be selected.

3. PROPOSED METHOD

This section describes our proposed method of solving MinMax 0/1 knapsack problem using dynamic programming. To describe the workings of the dynamic programming in finding the optimal solution of MinMax *0/1* knapsack, we use Example 1 below.

Example 1. Consider 4 items (*x1, x2, x3, x4*), each of which has a weight and value (*w1, w2, w3, w4*) = (3,4,2,2), (p1, *p2, p3, p4*) = (12, 14, 7, 6). If minimum capacity M_1 = 5 and the maximum capacity M_2 = 6, then some alternative solutions to these problems is shown in Table 1. In the table it can be seen that under the MinMax *0/1* knapsack problem it is alternative number 5, 6, 7, and 8 that meet the total weight of 5. Of the four alternatives, the optimal solution is simply an alternative number 6 with a total value of the item for 18 (minimum).

Table 1. Items that can be put into knapsack

Alternatives	Items that are taken	Total weight of items taken	Total value of items taken
1	x_1	3	12
2	x_2	4	14
3	x_3	2	7
4	x_4	2	6
5	x_1, x_3	3+2 = 5	12 + 7 = 19
6	x_1, x_4	3+2= 5	12+6 = 18
7	x_2, x_3	4 + 2 = 6	14 + 7 = 21
8	x_2, x_4	4 + 2 = 6	14 + 6 = 20
9	x_3, x_4	2 + 2 = 4	7 + 6 = 13

Finding optimal solution using Dynamic Programming contains several steps, namely [14]:

1. Determine the optimal solution's structure.
2. Recursively define the optimal solution.
3. Determine the optimal solution in forward or reverse.
4. Construct optimal solution.

Optimal solution of MinMax *0/1*knapsack problem can be determined by forward or backward approach. In this paper we use the backward approach. Our backward approach is based on the backward approach used to solve 0/1 knapsack problem, which has the recurrent equation below [15]:

$$f_j(y) = \max\{ f_{j-1}(y), f_{j-1}(y-w_j) + p_j\} \quad (3)$$

In this paper, *U* represents the total weight and *U [j, y]* represents the total weight of the *j* stage at a capacity of y. *L* represents an optimum total value of items and *L [j, y]* represents the optimum total value of items that are included in the knapsack at the *j* stage at a capacity of *y*.

Step 1: Determine Structure of the Optimal Solution

Integer knapsack with MinMax constraint using dynamic programming can be divided into two conditions:

1. Conditions at $w_j > y$.

The condition was stated that when item *j* has the weight greater than the capacity of *y* then the item is not included in the knapsack.

2. Condition at $w_j \leq y$.

The condition that stated that when the weight item w_j is less than or equal to the weight of the capacity of *y* the item could be included in the knapsack.

At the time the item is selected, the total weight of the total weight of the optimum value of the previous stage at the $y\text{-}w_j$ added with the weight of w_j or $U[i\text{-}1, y\text{-}w_j] + w_j$.

The value of $U[i\text{-}1, y\text{-}w_j]+w_j$ an also be called the stage at the time of *j*. The total value of at least the items will follow the results of the total weight of the optimal value of the minimum total value of the item is $L[i\text{-}1, y\text{-}w_j] + p_j$.

The optimum value of the total weight will be determined by four conditions, namely**:**

1) If the total weight less than the optimum value of the total weight of the previous stage then the item is not included in the knapsack.
   The optimum value of the total weight of stage j is taken from the optimum value of the total weight of the previous stage or *U[j-1,y]* so that the minimum total value of item j phase is also taken from a minimum total value of the item earlier stage or *L[j-1,y]*.
2) If the total weight greater than the optimum value of the total weight of the previous stage of the items can be inserted into the knapsack so that the optimum value of the total weight of a $U[j\text{-}1, y\text{-}w_j] + w_j$. Similarly, a minimum total value of the item will be $L[j\text{-}1, y\text{-}w_i] + p_j$.
3) If the total weight equal to the optimum value of the total weight of the previous stage then applied the principle of minimal value.
   a. If the total value of at least the previous phase or *L[j-1,w]* is smaller than the total value of at least stage j or $L[j\text{-}1, y\text{-}w_j] + p_i$ then the item is not included in the knapsack. Minimum total value is the total value of the previous stage or *L[j-1,w]*.
   b. If not then the item is placed in the knapsack so that the minimum total value of the item will be $L[i\text{-}1, y\text{-}w_j] + p_j$.
4) Conditions at the *U[j,y] ≥ MinCapacity*.
   If *U[j,y-1]≥MinCapacity* or *U[j-1,y]≥MinCapacity* the principle of minimal value for the total value of the item applied. Value of *L[j,y]* as compared to the value of *L[j,y-1]* and selected the smallest total value, this is due to minimizing the total value of the item at the minimum limit has been met. Furthermore *L[j,y]* dan *L[j-1,y]* re compared to select the smallest value. If the value of *L[j,y]=L[j,y-1]* atau *L[j,y]=L[j-1,y]* then will be selected value of the largest total weight of the item. It is tailored to the application of the loading of goods into the container in order to minimize the remaining container space.

Step 2: Determine Recursive Equations for MinMax 0/1 Integer Knapsack

Based on previous analysis the recursive equation can be formulated as follows:

$$\text{If } w_j > y \begin{cases} U[j,y] = U[j-1,y] \\ L[j,y] = L[j-1,y] \end{cases} \quad (4)$$

If $w_j \leq y$ then
If $U[j-1,y] \neq U[j-1,y-w_j]+w_j$ then
$U[j,y] = \textbf{Max}\ \{ U[j-1,y], U[j-1,y-w_j] + w_j\}$
If $U[j,y] \geq MinCapacity$ and $U[j,y-1] \geq MinCapacity$ then
$L[j,y] = \textbf{Min}\ \{ L[j,y-1], L[j,y] \}$
If $U[j,y] \geq MinCapacity$ and $U[j-1,y] \geq MinCapacity$ then
$L[j,y] = \textbf{Min}\ \{ L[j-1,y], L[j,y] \}$

The base of the recursive equation (4) is when $w_j=0$ or $y=0$ and the values of $U[j,y]= 0$ and $L[j,y]= 0$.

.

Step 3: Determine Optimal Solutions

In Table 2 the initial stage is the stage where the item has not been chosen so that $U[0,y]=0$ and $L[0,y]=0$ for all values of y. When y=1 and y=2, the value of $w_j>y$ so that the value of $U[j,y] = U[j-1,y]$ or optimum value is the total weight of the optimum value of the total weight of the previous stage. Similarly, the minimum total value of items with a minimum total value of items the previous stage or $L[j,y] = L[j-1,y]$. *Item* x_1 contained in y=3, y=4, y=5 and the previous phase is 0 so the solution is still optimal for a minimum total value of items is 12 and the total weight of the optimal solution for a maximum of 3.

At $y=4$, *item* x_2 can be incorporated into the knapsack item with value of 14. At these conditions, the total weight of the previous stage is not equal to the total weight of the stage 2 or $U[j-1,y] \neq U[j-1,y-w_j] + w_j$ thus selected that the maximum between the two. Value of $U[j-1,y-w_j] + w_j$ is greater than the value of $U[j-1,y]$ thus that value becomes the optimum solution when y=4 and has not met the requirements of $U[j,y] \geq MinCapacity$.

At y=5 and y=6 satisfy the condition $U[j,y] \geq MinCapacity$ but the $U[j,y-1]$ and $U[j-1,y]$ do not meet the requirements of greater than *MinCapacity*. Minimum total value of items adapted to the results that have been selected by the $U[2,4]$ (the total weight at the time of y = 4).
At this stage the value used is $U[j-1,y-w_j] + w_j$ (at step 2) thus the value of $L[j,y] = L[j-1,y-w_j] + p_j$.
At y=5 with j=3 or item 3, the value of $U[j-1,y-w_j] + w_j = 5$ is greater than the value of $U[j-1,y] = 4$ so the value of $U[j-1,y-w_j] + w_j$ will be the optimal solution when y=5. The value of $U[j,y-1]$ and $U[j-1,y]$ do not meet the requirements of greater than *MinCapacity* so $L[j,y]$ will be taken at y=5 or $L[j-1,y-w_j]+ p_j$. At y=6, *the value of* $U[j-1,y-w_j]+w_j$ *is greater than the value of* $U[j,y-1]$ thus the value of $L[j,y] = L[j-1,y-w_j]+p_j$. The value of $U[j,y-1]$ satisfy the requirements of KapasitasMin than the total value of the determination of minimal solutions of recursive equation $L[j,y]=Min\{ L[j,y-1], L[j,y]\}$ do value $U[j,y]$ at U=5 and L=19.

The final stage when y=6 and item 4 is the minimum total value of the solution. Recursive equation $L[j,y]=Min\{L[j,y-1],L[j,y]$ } produces a value of 18. Value of $U[j,y]$ to adjust the value of the ratio between $L[j-1,y]$ *and* $L[j,y]$. Furthermore the condition $U[j-1,y] \geq MinCapacity$ met so enacted $L[j,y]=Min\{L[j,y],L[j-1,y]$ }. Minimum total value is taken at y=6 fixed $L[j,y]$ at 18 so total value of the maximum weight is also fixed to a value of 5. Optimal solution of Example 1 on the reverse approach is *fn(M)* [7] or at the time $L[n,c]$ and $U[n,c]$ and demonstrate the value of $L[n,c]$ is 18 and the value of $U[n,c]$ is 5.

Procedure *CalculationofOptimalSolution(MinCapacity,MaxCapacity,n,W,P,U,L)*

/ / Input MinCapacity and MaxCapacity are minimal and maximal limits total
/ / weight of item knapsack. *N* is the number of items that is, *W* is the array of data
/ / weight of each item, *P* is the array of data item values, *U* is the optimal solution the data array
/ / total weight of item *j* at the stage to a capacity of *y* and *L* is the array of data solutions
/ / optimal total value of item *j* at the current stage of capacity *y*.

For *y←0 to MaxCapacity* ***do***
$L[0,y]\leftarrow 0;\ U[0,y]\leftarrow 0;$

For *j←1 to MaxCapacity* ***do***
$L[j,0]\leftarrow 0;U[j,0]\leftarrow 0;$

```
For y←1 to MaxCapacity do
Begin
        If W[j] <=y then
                If W[j] + U[j-1, y-W[j]]≠U[j-1,y] then
                        U[j,y]←Max{ W[j] + U[j-1, y-W[j]],U[j-1,y]};
                Else   L[j,y]←Min {P[j]+L[j-1,y-W[j], L[j-1,y] };
        Else U[j,y]
             L[j,y];
        If U[j,y] ≥ MinCapacity and U[j,y-1] ≥ MinCapacity
                L[j,y] ← Min { L[j,y-1], L[j,y] };
        If U[j,y] ≥ MinCapacity and U[j,y-1] ≥ MinCapacity
                L[j,y] ← Min { L[j,y], L[j,y-1] };
End
```

Table 2 Calculation of optimal solution of *MinMax 0/1 knapsack*

j/y	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	U=0 L=0 (0,0,0,0)	U=0 L=0 (0,0,0,0)	U=3 L=12 (1,0,0,0)	U=3 L=12 (1,0,0,0)	U=3 L=12 (1,0,0,0)	U=3 L=12 (1,0,0,0)
2	0	U=0 L=0 (0,0,0,0)	U=0 L=0 (0,0,0,0)	U=3 L=12 (1,0,0,0)	U=4 L=14 (0,1,0,0)	U=4 L=14 (0,1,0,0)	U=4 L=14 (0,1,0,0)
3	0	U=0 L=0 (0,0,0,0)	U=2 L=7 (0,0,1,0)	U=3 L=12 (1,0,0,0)	U=4 L=14 (0,1,0,0)	U=5 L=19 (1,0,1,0)	U=5 L=19 (1,0,1,0)
4	0	U=0 L=0 (0,0,0,0)	U=2 L=6 (0,0,0,1)	U=3 L=12 (1,0,0,0)	U=4 L=13 (0,0,1,1)	U=5 L=18 (1,0,0,1)	U=5 L=18 (1,0,0,1)

Step 4: Construct the Optimal Solution of MinMax 0/1 Knapsack

Recursive equation (3) showed only minimal items and the total value of the total weight of the item, but did not show any items selected for inclusion into the knapsack according to the total value of at least 18. Final stages of completion with the construction of dynamic programming is the optimal solution in which it traces the stages of the sequence of items selected from the phase of *U [n, c]* to the first stage or *U[0,0]*.

These measures include selected search fields:

1. Search of the *U[j,y]* to *U[j,y-1]* until there is a change in value between two or *L[j,y]* with *L[j,y-1]* (horizontally), then the item if there is a change in *j* as *y* phase included into the knapsack and the second step search is then performed.
2. Search of the *U[j,y]* the results of the first step compared to the previous stage *U[j-1,y]* or *L[j,y]* with *L[j-1,y]* (vertically), in case of change in value then the item is an item selected and continued with the remaining capacity of the selected item or *y-wj*. Then searched for back to step 1 to $y = 1$ or $j = 0$.

Based on Table 2, the construction steps of the optimal solution of Example 1can be described as follows:

1. Optimal solution value at $y = 6$ and $j = 4$ is the optimal value for all stages $j = 4$, a search using a step change in value when $y = 5$.
2. The next step to be checked with the previous phase 2 is *U[j-1,y]*. Changes occur in the value of *L[3,5]*=19 is not equal to *L[4,5]*=18 so item 4 included into the knapsack.
3. Residual capacity become 3 so the search starts at *U[3,3]*.
   Step 1 produces $j = 3$ and $y = 3$ and then proceed step 2. Search ends at $j = 1$ and $y = 3$ so first item included into the knapsack. Residual capacity to 0 then the optimal solution construction ends.

Procedure *OptimalSolutionConstruction (MaxCapacity,n,U,L,W)*

```
/ / Input MaxCapacity is the maximum total weight limit knapsack item. N is the number of items
available, W is the array of data
/ / weight of each item, U is a data array
/ / optimal solution to the total weight of item j at the current stage of capacity y and L is the array of
data total value of items the optimal solution stage of j at a capacity of y.

x←0; j←n; y←MaxCapacity;
while (j>0) and (y>0) do
    If U[j,y] = U[j,y-1] then
            y←y-1;
    Else
            j←j-1;
            if U[j,y] <> U[j-1,y] or L[j,y] <> L[j-1,y] then
            Begin
                    Display item of j;
                    y←y-W[j];
            End;
end while
end;
```

4. EXPERIMENTAL RESULTS

In this section we apply the solution of MinMax 0/1 knapsack problem using dynamic programming to the problem of loading goods into the container to optimize the available container space. Minimum and maximum data capacity or volume of available container space is shown in Table 3.

The data used in this experiment is received from PT DFI whose business is shipping containers to Singapore. Each container sent has a minimum volume limit. If the total volume of the container is less than the minimum limit, the company will be fined per cubic meter. Data items in the PT DFI is an item of data collected on March 5, 2010 and March 15, 2010, the data item is delivered in stages.The data is entered into by the company's container measuring 20 'and 40' respectively and as many as two containers are shown in Table 4. Application of the integer knapsack with minmax constraint is done by filling the first container size of 20 'where the calculated optimal solution with minimal restrictions produce 30 m3 capacity of 30 589 m3 optimal solution with minimal total weight of 4905.30 kg and are shown in Table 5. These results do not exceed the actual maximum capacity of 31 152 m3 and a maximum weight of 20 ton containers.

The second container using a size 20 ', this is because the item was left sufficient for the container size of 20' and the data items used are the remaining items of data from the first kotainer. Items selected were all the rest of the stuff from the first container. The results of the second container loading for the rest there are a lot more space so if there is an additional delivery of goods can be done to maximize the container space and shown in Table 5.

If data on March 5, 2010 using a container size of 40 ' then it will fill the space of 49.931 m3 of container and the minimum total weight of selected items of 14838.30 kg and all the selected items are shown in Table 6. Furthermore, the data item to the date of March 15, 2010 will be the first time using a 20 'container with the results shown in Table 7. The second container for goods dated March 15, 2010 performed by using the container size of 40 'will be but there is one item that still remains. If all the data on 5 and March 15, 2010 were collected then generates an optimal solution that uses only 3 containers and are shown in Table 8 so it is more efficient than delivery by PT DFI which uses four containers.

Table 3 Container data

No	Container Length	Volume	Maximal weight	Minimal volume
1	20 *feet*	31.152 m^3	20 ton	20 m^3
2	40 *feet*	62.683 m^3	30 ton	40 m^3

Table 4 Data of PT DFI

No	Container	Date	Capacity	Total weight
1	20	March 5, 2010	23.661 m^3	8,222.50 kg
2	20	March 5, 2010	26.270 m^3	6.615,80 kg
3	40	March 15, 2010	45.999 m^3	13,987.40 kg
4	40	March 15, 2010	47.469 m^3	7,832.20 kg

Table 5 Data of 5th March 2010 using ***MinMax 0/1 knapsack***

No	Container	Date	Capacity	Total Weight
1	20	March 5, 2010	30.589 m^3	4905.80 kg
2	20	March 5, 2010	19.342 m^3	9932.50 kg

Table 6 Data of 5th March 2010 using ***MinMax 0/1 knapsack***

No	Container	Date	Capacity	Total Weight
1	40	March 5, 2010	49.931 m^3	14838.30 kg

Table 7 Data of 15th March 2010 using ***MinMax 0/1 knapsack***

No	Container	Tanggal	Capacity	Total Weight
1	20	March 15, 2010	30.535 m^3	4744 kg
2	40	March 15, 2010	61.933 m^3	16875.60 kg

Table 8 Data of 5th March 2010 (use all data) using *MinMax 0/1 knapsack*

No	Container	Tanggal	Capacity	Total Weight
1	40	March 5 & 15, 2010	62.117 m^3	19244.30 kg
2	40	March 5 & 15, 2010	60.691 m^3	9858 kg
3	20	March 5 & 15, 2010	21.215 m^3	7555.60 kg

5. CONCLUSIONS

Based on the discussion in previous chapters, we conclude that:

1. MinMax 0/1 knapsack can be solved using dynamic programming so that the total value of items is optimal (in this case minimal) while a minimum limit requirement is met without exceeding the maximum capacity limit.
2. MinMax 0/1 knapsack problem can be applied to the problem of loading of goods into the container so that the total weight is minimum and at the same time the minimum capacity requirement of container is met without exceeding the maximum capacity of the containers.

REFERENCES

[1] Bassard, G. and Bratley, P.;,"Fundamentals of Algorithmic", *Prentice Hall Inc*,1996
[2] Pisinger, D.;,"Algorithms for Knapsack Problems", *Ph.D. Thesis, DIKU Unversity of Copenhagen*, 1995
[3] Wilbout, C., Hanafi, S. and Salhi, S.;, "A Survey of Effective Heuristic and Their Application to a Variety of Knapsack Problems", *IMA Journal of Management Mathematics*,pp.227 – 244, 2008

[4] Yamada, T. and Kataoka, S.;, "Heuristic and Exact Algorithms for the Disjunctively Constraint Knapsack Problem", *Transactions of Information Processing Society of Japan*,vol.43, no. 9, pp. 2864-2870, 2002
[5] Yamada, T. and Futukawa, M.;, "Heuristic and Reduction Algorithms for the Knapsack Sharing Problem", *Computer Operation Research*, vol. 24, pp.961-967, 1997
[6] Gallo, G., Hammer, P.L. and Simeone, B;, "Quadratic Knapsack Problems", *MathematicalPrograming Studies*, vol. 12, pp.132-149, 1980
[7] Yu, G.;, "On the max-min 0-1 Knapsack Problem with Robust Optimization Application",*Operation Research*,44, pp. 407-415, 1996
[8] Iida, H.;, "On Solving the Max-Min 0-1 knapsack Problem", *Research Report Journal*, vol. IS-RR-97-0025F, pp.1-23, 1997
[9] Lagoudakis, M. G.;, "The 0-1 Knapsack Problem: An Introductory Survey", *Technical Report, The Center for Advanced Computer Studies,University of Southwestern Louisiana*, pp.8-9, 1996
[10] Rolfe, J. T.;, "An Alternative Dynamic Programming Solution for the 0/1 Knapsack", *ACM SIGCSE bulletin*, vol. 39, pp.54-56, 2007
[11] Brotcorne, L., Hanafi, S. and Mansi R.;,"A dynamic programming algorithm for the bilevel knapsack problem", *Operations Research Letters*, vol. 37, no. 3, pp. 215–8, 2009
[12] Boyer, V., El Baz, D. and Elkihel, M.;, "A dynamic programming method with lists for the knapsack sharing problem". *Computers & Industrial Engineering*, vol. 61, no 2, pp. 274–8, 2011
[13] Rong, A., Figueira J.R. and Klamroth, K.;, "Dynamic programming based algorithms for the discounted {0–1} knapsack problem", *Applied Mathematics and Computation* , vol. 218, no. 12, pp. 6921–33, 2012
[14] Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C.;,"Introduction to Algorithms", *Massachusetts Institute of Technology Press, 2nd Edition*, 2001
[15] Horowitz, E. and Sahni, S.;,"Fundamentals of Computer Algorithms", *Computer Science Press Maryland*, 1988

BIOGRAPHY OF AUTHORS

Ani Dijah Rahajoe received her bachelor degree in Information Engineering from Surabaya University, Indonesia, M.Cs in Computer Sciences from Universitas Gadjah Mada, Indonesia. She currently works as lecturer at Department of Engineering, Faculty of Information Engineering, Bhayangkara University, Indonesia. Her research interests are optimalization algorithm, data warehousing and data mining.

Edi Winarko received his bachelor degree in Statistics from Universitas Gadjah Mada, Indonesia, M.Sc in Computer Sciences from Queen University, Canada, and Ph.D in Computer Sciences from Flinders University, Australia. He currently works as lecturer at Department of Computer Sciences and Electronics, Faculty of Mathematics and Natural Sciences, Universitas Gadjah Mada. His research interests are data warehousing, data mining, and information retrieval. He is a member of ACM and IEEE.