Yugoslav Journal of Operations Research
Vol 19 (2009), Number 1, 113-122
DOI:10.2298/YUJOR0901113I

REALIZATION OF KNAPSACK PROBLEM SOLVING ALGORITHM AND SOME OF ITS APPLICATIONS

Dimiter IVANCHEV
New Bulgarian University, Sofia
divanchev@nbu.bg

Elena RADOVANOVA
Technical University of Sofia
ear@abv.bg

Received: December 2007 / Accepted: May 2009

Abstract: A realization of one algorithm for solving the classical knapsack problem which is much faster than the dynamical programming method and requires less memory is suggested. The popular situation in the Big Brother TV show is used to exemplify its applicability. For the purpose, the number of the wishes of all players are maximized.

Keywords: Knapsack problem, algorithmic realization, optimum budget, network optimization.

1. INTRODUCTION AND MOTIVATION

The following situation is frequently observed. A group of persons try to arrange a budget (some quantity of money) for a future period – a week, a month, a year etc. Sometimes the interests of the persons are contradictable, which implies a different behavior of the players. Due to the large number of combinations a person of the group can be more or less satisfied.

A typical example of this situation is the *Big Brother* TV show. The players frequently have to arrange their budget for the next week. As a rule, they quarrel and feel dissatisfied.

Further, the above situation is taken as an example and some variants of the budget arrangement are suggested in order to maximize the number of wishes satisfied or to be sounder and fairer.

2. PROBLEM STATEMENT AND THE MODEL

Let us now consider the situation mentioned above and let us introduce the following notations:

K - the amount of money (currency), i.e. EUR we can use next week,

m - the number of all the players in the show.

n - the number of the products (items) they can order,

k_j - the price of the $j-$th item they can order and by, j=1,2,...,n.

Obviously, all numbers K, m, n *and* k_j are positive integers.

Further we set a_{ij} to be one, if the $i-th$ player wants to buy (order) the $j-th$ item and zero otherwise. Now we denote by $c_j := \sum_{i=1}^{m} a_{ij}$ (1) and this is the number of all the players who ordered the $j-th$ item, j=1,...,n.

Definition 1. *A vector* $x=(x_1,x_2,...,x_n)$ *is called a* $K-budget$, *if* $x_j \in \{0,1\}$, j=1,...,n *and* $\sum_{j=1}^{n} k_j x_j \le K$.

We denote by KB the set of all $K-budgets$.

The question is: how to maximize the number of the satisfied wishes of the group?

Now we can formulate a reasonable optimization problem:

$$P: \quad \max\left\{\sum_{i=1}^{n} c_i x_i / s.t.\ x \in KB\right\} .$$

P means to find a $K-budget\ x$, which maximizes the number of the satisfied wishes of the whole group of players.

3. SOLUTION METHODS

P is a knapsack problem ([2],[3],[4]). It can be solved using a Dynamic Programming Method (DPM) or the Critical Path Method (CPM) in a PERT-like network.

First variant (DPM). We define the *Bellmann's* functions

$$B_l(k) := \max\left\{\sum_{j=1}^{l} c_j x_j / \ s.t. \sum_{j=1}^{l} k_j x_j \le k,\ x_j \in \{0,1\}\right\}$$

for $k=0,1,2,...,K$ and $l=1,2,3,...,n$ and find the recurrent equality (3) for all $k=0,1,2,...,K$ and $l=2,3,...,n$:

$$B_l(k) := \max\left\{c_l x_l + B_{l-1}(k - k_l x_l) \;/\; s.t.\; 0 \le k_l x_l \le k,\; x_l \in \{0,1\}\right\} \\ = c_l \hat{x}_l(k) + B_{l-1}(k - k_l \hat{x}_l(k)). \tag{2}$$

The optimum solution is a vector $\hat{x} \in KB$ with the coordinates

$\hat{x}_n := \hat{x}_n(K)$,

$\hat{x}_{n-1} := \hat{x}_{n-1}(K - k_n \hat{x}_n)$,

$\hat{x}_{n-2} := \hat{x}_{n-2}(K - k_n \hat{x}_n - k_{n-1}\hat{x}_{n-1})$,

$\hat{x}_2 := \hat{x}_2(K - k_n \hat{x}_n - k_{n-1}\hat{x}_{n-1} - \ldots - k_3 \hat{x}_3)$,

$\hat{x}_1 := \hat{x}_1(K - k_n \hat{x}_n - k_{n-1}\hat{x}_{n-1} - \ldots - k_2 \hat{x}_2)$,

which maximizes the number of the satisfied wishes to $B_n(K) = \sum_{j=1}^{n} c_j \hat{x}_j$.

Second variant (CPM) [1]**.** We define a PERT network $G = (V, A)$ with node set V, arc set A and arc weights as follows: all nodes from V are integer points (i, j) with $0 \le i \le K$, $0 \le j \le n+1$.

On level 0 there is only one vertex $V_1 = (0,0)$.

Let $(i, j-1)$ be an arbitrary vertex from level $j-1$, $j = 1,2,\ldots,n$. We define two arcs starting from node $(i, j-1)$ and ending at (i, j) and at $(i + k_j, j)$, if $i + k_j \le K$. If the last inequality is not fulfilled we define the first arc only. We set for the first arc length 0 and for the second one length c_j. Finally we connect all the arcs from level n with node $(K, n+1)$ and we set for all of them an arc length 0.

P is equivalent to find the longest path (critical path) in this PERT network from point $(0,0)$ to point $(K, n+1)$. For all the arcs from type $((i, j),(i, j+1))$ from this path we set $x_{j+1} := 0$, for the others (they are from type $((i, j),(i + k_j, j+1))$ we set $x_{j+1} := 1$. Now, $x = (x_1, x_2, \ldots, x_n)$ is an optimum $K - budget$ ([1]).

This network can be formed as follows. We define three arrays VL_j (Vertex-Level of V_j), SWV_j (Support-Weight-Vertex of V_j) and IVL_q (Index- Vertex-Level q) with the meaning:

j - number of vertex $V_j = (p, q)$,

$VL_j = p$ - position of this vertex on level q,

$|SWV_j|$ - length of the longest path from $(0,0)$ to (p, q); if $SWV_j \ge 0$ then the last vertex before (p, q) is $(p, q-1)$ else it is $(p - k_q, q-1)$,

IVL_q - position in VL and SWL where the information for all nodes on level q starts,

$q = 1,2,...,n, \quad p = 0,1,...,K.$

Now, let $V_j = (p,q)$ belong to the longest path from $(0,0)$ to $(K,n+1)$. The last vertex before it from this longest path is $(p,q-1)$ if $SWV_j \geq 0$ and it is $(p-k_q,q-1)$ otherwise. The length of the longest path from $(0,0)$ to (p,q) is $|SWV_j|$. We define x_q as one or zero if SWV_q is negative or nonnegative.

Without the loss of generality we assume $k_i \leq k_j$ for all $i,j = 1,2,...,n,\ i < j$.

Further we suggest the following algorithm using a PASCAL - like language.

4. ALGORITHM

0^0. (* Input: $c = (c_1,...,c_n)$ with $c_i > 0$,
$k = (k_1,...,k_n)$, sorted as the above mentioned
Output: $\hat{x} = (\hat{x}_1,\hat{x}_2,...,\hat{x}_n)$ - solution of the problem *)

1^0. (* Initialization *)

$VL_1 := 0; \quad VL_2 := k_1; \quad SWV_1 := 0; \quad SWV_2 := -c_1; \quad IVL_1 := 1; \quad IVL_2 := 2;$

2^0. (* Main Loop – forming the $j-th$ level *)

for $j := 1$ *to n do*

begin

$IVL_{j+1} := IVL_j + 1; \quad l := IVL_j - 1;$

for $i := IVL_{j-1}$ *to* $IVL_j - 1$ *do*

begin $l := l+1; VL_l := VL_i; SWV_l := |SWV_i|$ *end*;

for $i := IVL_{j-1}$ *to* $IVL_j - 1$ *do*

begin if $VL_i + k_j = p$ *for some node*

$V_l = (p, j-1)$ *with* $l \in [IVL_{j-1}, IVL_j - 1]$

and $|SWV_i + c_j| > |SWV_l|$ *then*

$SWV_l := -SWV_i - c_j$

else

begin

$l := IVL_{j+1}; VL_l := j;$
$SWV_l := -|SWV_i| - c_j; IVL_{j+1} := l$

end

end

end (* of Main Loop *)
3^0. (* Finding of the solution $\hat{x}$ *)
for $j := IVL_n$ *to* $IVL_{n+1} - 1$ *do*

find the max*imum* $|SWV_j|$, *let it be* $|SWV_p|$;

for $j := n$ *downto* 1 *do*

begin

if $SWV_p \geq 0$ *then*

begin $\hat{x}_j := 0;$ *find node* $V_l = (p, j-1)$ *end*

else

begin $\hat{x}_j := 1;$ *find node* $V_l = (VL_p - k_j, j-1)$ *end*

$p := l$

end. (* of algorithm *)

Justification of the algorithm. First we define the nodes from level 1 – $V_1 = (0,1)$ and $V_2 = (k_1,1)$. Then – from level 2, they are $V_3 = (0,2)$, $V_4 = (k_1,2)$, $V_5 = (2k_1,2)$ if $k_1 = k_2$ or $V_3 = (0,2)$, $V_4 = (k_1,2)$, $V_5 = (k_2,2)$, $V_6 = (k_1 + k_2,2)$ if $k_1 < k_2$ etc. to the level n. If we keep the rule $k_i \leq k_{i+1}, i = 1,2,...,n-1$ then $V_i = (p,q), V_{i+1} = (r,q) \Rightarrow p < r$ is fulfilled for all $i = IVL_q,...,IVL_{q+1} - 1$ from the level q. On the other hand this rule minimizes the number of nodes in the network. We need the information for the nodes of the network only. For each node $V_i = (p,q)$ we need the information for $(p,q-1)$ and $(p-k_q,q-1)$ in order to calculate the length of the longest path from $(0,0)$ to $V_i = (p,q)$ - this is SWV_i.

5. ILLUSTRATIVE EXAMPLE

We have considered a situation with $m = 6$ players, $n = 5$ items and amount of money at $K = 10$ units. The matrix $A = (a_{ij})_{m\,x\,n}$ with the players' wishes is given below:

$$A = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix}.$$

All the prices $k_1,...,k_5$ of the 5 items and all the weights $c_1,...,c_5$ from (1) are given in Table 1.

j	1	2	3	4	5
k_j	3	2	1	2	4
c_j	4	2	5	4	3

Table 1

The first variant (DPM).

All the *Bellmann's* functions are given as:

$$B_1(k) := \max\{4x_1 / 0 \le 3x_1 \le k,\ x_1 \in \{0,1\}\},$$

$$B_2(k) := \max\{2x_2 + B_1(k-2x_2) / 0 \le 2x_2 \le k,\ x_2 \in \{0,1\}\},$$

$$B_3(k) := \max\{5x_3 + B_2(k-x_3) / 0 \le x_3 \le k,\ x_3 \in \{0,1\}\},$$

$$B_4(k) := \max\{4x_4 + B_3(k-2x_4) / 0 \le 2x_4 \le k,\ x_4 \in \{0,1\}\},$$

$$B_5(k) := \max\{3x_5 + B_4(k-4x_5) / 0 \le 4x_5 \le k,\ x_5 \in \{0,1\}\}.$$

All the function values $B_l(k)$ with all the optimum points $\hat{x}_l(k)$ are given in Table 2.

Finally we have got the optimum solution as follows. The maximum of the goal function of P_1 is $\sum_{j=1}^{5} c_j \hat{x}_j = B_5(10) = 16$ satisfied wishes. The optimum solution is:

$\hat{x}_5 = \hat{x}_5(10) = 1,$

$\hat{x}_4 = \hat{x}_4(10-k_5\hat{x}_5) = \hat{x}_4(10-4.1) = \hat{x}_4(6) = 1,$

$\hat{x}_3 = \hat{x}_3(10-k_5\hat{x}_5 - k_4\hat{x}) = \hat{x}_3(10-4.1-2.1) = \hat{x}_3(4) = 1,$

$\hat{x}_2 = \hat{x}_2(10-k_5\hat{x}_5 - k_4\hat{x}_4 - k_3\hat{x}_3) = \hat{x}_2(10-4.1-2.1-1.1) = \hat{x}_2(3) = 0,$

$\hat{x}_1 = \hat{x}_1(10-k_5\hat{x}_5 - k_4\hat{x}_4 - k_3\hat{x}_3 - k_2\hat{x}_2) = \hat{x}_1(10-4.1-2.1-1.1-2.0) = \hat{x}_1(3) = 1.$

Hence, the optimum budget is $\hat{x} = (1,0,1,1,1)$, i.e. they have to order all the items excluding the second one only and will pay the whole amount of money $K = 10.$

Table 2

k	0	1	2	3	4	5	6	7	8	9	10
$B_1(k)$	0	0	0	4	4	4	4	4	4	4	4
$\hat{x}_2(k)$	0	0	0	1	1	1	1	1	1	1	1
$B_2(k)$	0	0	2	4	4	6	6	6	6	6	6
$\hat{x}_2(k)$	0	0	1	0	0	1	1	1	1	1	1
$B_3(k)$	0	5	5	7	9	9	11	11	11	11	11
$\hat{x}_3(k)$	0	1	1	1	1	1	1	1	1	1	1
$B_4(k)$	0	0	5	9	9	11	12	12	15	15	15
$\hat{x}_4(k)$	0	0	0	1	0/1	1	1	1	1	1	1
$B_5(k)$	0	0	0	0	9	11	13	13	15	15	16
$\hat{x}_5(k)$	0	0	0	0	0	0	0	0	0	0	1

It is possible to find all the optimum budgets from Table 2 for all $K \le 10.$

For $K = 8$ we have found the optimum budget as $\hat{x} = (1,1,1,1,0)$, which means to buy all the items without the last one. The number of the wishes satisfied is $B_5(8) = 15.$

For $K = 5$ we have found the optimum solution as $\hat{x} = (0,1,1,1,0)$ and $B_5(5) = 11.$

The second variant (CPM).

We have considered the same example and enumerated the variables according to the substitution $y = (y_1, y_2, y_3, y_4, y_5) = (x_3, x_2, x_4, x_1, x_5)$ in order to keep the

sufficient condition fulfilled. Further, we have calculated the data in Table 3 and Table 4 according to our algorithm.

Since $\max\left\{\left|SWV_j\right| / j\right\} = \left|SWV_{32}\right| = \left|-16\right|$ and $SWV_{32} < 0$,then $y_5 = 1 \quad (= \hat{x}_5)$.

Further we have calculated $q = K - k_5 = 10 - 4 = 6$ and have found a node with $VL_p = 6$ from the level 4 – it is for $p = 19$. Since $SWV_{19} = -13 < 0$ then $y_4 = 1\,(= \hat{x}_1)$. Then: $q = K - k_5 - k_4 = 10 - 4 - 3 = 3$; the node from the level 3 with $VL_p = 3$ is for $p = 10$, since $SWV_{10} = -9 < 0$ then $y_3 = 1\,(= \hat{x}_4)$;

– $q = K - k_5 - k_4 - k_3 = 10 - 4 - 3 - 2 = 1$, the node from the level 2 with $VL_p = 1$ is for

$p = 4$, since $SWV_4 = 5 \geq 0$ then $y_2 = 0\,(= \hat{x}_2)$;

– $q = K - k_5 - k_4 - k_3 = 10 - 4 - 3 - 2 = 1$, the node from the level 1 with $VL_p = 1$ is for

$p = 2$, since $SWV_2 = -5 < 0$ then $y_1 = 1\,(= \hat{x}_3)$.

Hence, $\hat{x} = (1,0,1,1,1)$.

Table 3

a)

	Level 1		Level 2				Level 3					
j	1	2	3	4	5	6	7	8	9	10	11	12
VL_j	0	**1**	0	**1**	2	3	0	1	2	**3**	4	5
SWV_j	0	**-5**	0	**5**	-2	-7	0	5	-4	**-9**	-6	-11

b)

	Level 4								
j	13	14	15	16	17	18	19	20	21
VL_j	0	1	2	3	4	5	**6**	7	8
SWV_j	0	5	-4	9	-9	11	**-13**	-10	-15

c)

	Level 5										
j	22	23	24	25	26	27	28	29	30	31	32
VL_j	0	1	2	3	4	5	6	7	8	9	**10**
SWV_j	0	5	4	9	9	11	13	-12	15	-14	**-16**

Table 4

j	1	2	3	4	5	6
IVL_j	1	3	7	13	22	33

The geometrical visualization is given in Figure 1. The optimum solution with length 16 is marked with a broken line. All the others with length (from right to left) 14, 15, 12, 13, 11, 9, 9, 4, 5 and 0 are given with a hard line.

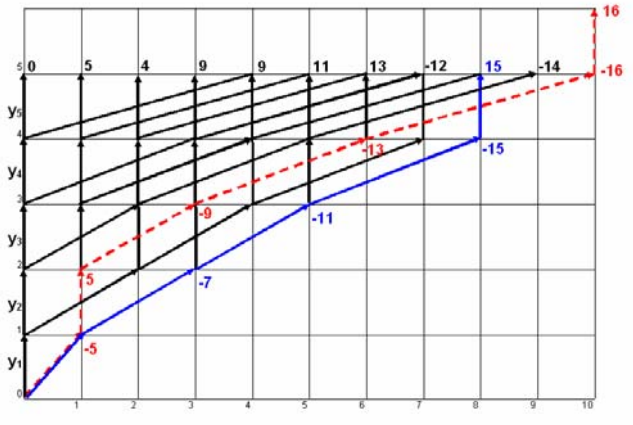

Figure 1

6. CONCLUSION

The problem can be generalized in different directions or formulated in another way, i.e. not to ask whether to buy or not to buy the $j-th$ item, but to find how many items from this type to buy. The only change in the algorithm is to drop out the restriction $x \in \{0,1\}$ from (3).

If we compare both methods suggested in Section 3 we can see that the *Bellmann's* Table 2 is to sparse (to many zeroes and equal elements). It is possible to reduce the memory needed if we use the second variant – CPM in an appropriate way, given in the previous section. In our example we have stored 110 integers using DPM. CPM needs 71 integers to be stored. That is 35% less.

The time complexity of both these methods is polynomial $O(Kn)$ nevertheless the knapsack problem is NP-hard one in a general case.

REFERENCES

[1] Ivanchev, D., *Mathematical Methods for Network Optimization*, NBU, Sofia, 2007.
[2] Gessner, P., and Wacker, H., *Dynamische Optimierung*, Carl Hansen -Verlag, München, 1972.
[3] Terno, J., „Numerische Verfahren der diskreten Optimierung (Modelle-Algorithmen-Komplexität)“, *Teubnet-texteE zur Mathematik*, Band 36 (1981) 168.
[4] Korte, B., J.,Vygen, *Combinatorial Optimization (Theory and Algorithms)*, Springer, Berlin, Heidelberg,New,York, 2000.