

HEURISTIC AND REDUCTION ALGORITHMS FOR THE
KNAPSACK SHARING PROBLEM

Takeo Yamada†‡ and Mayumi Futakawa§

Department of Computer Science, The National Defense Academy, Yokosuka, Kanagawa 239, Japan

(Received June 1996; in revised form January 1997)

Scope and Purpose—In the knapsack problem (KP) we are given a knapsack of a fixed capacity and a set of items with weight and value associated with each of them. The problem is to find the subset of items to be included in the knapsack such that the total value of the items included is as large as possible. KP is fundamental in combinatorial optimization and numerous articles were devoted to this subject.

In the current article we present a novel variation of the problem where each item belongs to one of two or more owners. Owners wish to make the total value of their own items in the knapsack as large as possible, irrespective to the contents of other owners. A reasonable solution in such a case is to maximize the minimum level of the value that can be guaranteed to all the owners. Thus we formulate a knapsack-type problem with a max-min objective function, and call this the knapsack sharing problem (KSP). Approximate and exact solution algorithms are well explored for KP, as well as the methods to reduce the size of the problem. We develop similar algorithms for KSP, and present a heuristic and reduction algorithms together with the results of computational experiments.

Abstract—KSP is newly formulated as an extension of the knapsack problem. KSP is \mathcal{NP} -hard, and we give a fast algorithm to find an upper bound to this problem. Based on this we develop a heuristic algorithm to solve KSP approximately. Also, a pegging test is derived that enables us to reduce the size of the original problem. Computational experiments are carried out to examine the behavior of the developed algorithms for several instance types of different statistical characteristics. © 1997 Elsevier Science Ltd

1. INTRODUCTION

The knapsack problem (KP) is fundamental in combinatorial optimization and literature abounds on this and related subjects (see, e.g. Martello and Toth [8]). In this article we discuss the following variation of the problem: we are given a set of items $N = \{1, 2, \dots, n\}$ with weight w_j and value p_j associated with each item $j \in N$. Every item belongs to one and only one of r disjoint classes N_1, N_2, \dots, N_r where:

$$\bigcup_{k=1}^r N_k = N, \quad N_k \cap N_l = \emptyset, \quad (k \neq l), \quad (1)$$

Also given is a knapsack of capacity \bar{c} . As in the ordinary (0-1) KP we wish to determine a set of items to be included in the knapsack. Thus, we introduce a vector $x = (x_j)$ of the following meaning: $x_j = 1$ if item j is selected, and $x_j = 0$ otherwise.

Our purpose is to maximize the minimum of $\{ \sum_{j \in N_k} p_j x_j \mid k = 1, 2, \dots, r \}$ under the ordinary capacity constraint, where $\sum_{j \in N_k} p_j x_j$ represents the total value of the k th class items in the knapsack. Therefore, mathematically the knapsack sharing problem (KSP) is formulated as;

$$\begin{aligned} \text{KSP: maximize} \quad & \min_{1 \leq k \leq r} \left\{ \sum_{j \in N_k} p_j x_j \right\} \\ \text{subject to} \quad & \sum_{j \in N} w_j x_j \leq \bar{c}, \\ & x_j \in \{0, 1\}, \quad j \in N. \end{aligned}$$

KSP is \mathcal{NP} -hard, since for $r=1$ the problem reduces to KP which is already \mathcal{NP} -hard.

† To whom all correspondence should be addressed (email: yamada@cs.nda.ac.jp).

‡ Takeo Yamada is a Professor at the National Defense Academy, Japan. He received his B.S. degree from Kyoto University and an M.S. and Ph.D. degrees from Stanford University. His current research interests include mathematical programming, dynamic systems and mathematical systems analysis in general.

§ Lieutenant Mayumi Futakawa received her B.S. degree from Kochi University, and an M.S. degree from the National Defense Academy. Her research interest is in combinatorial optimization and its application to communication security. Currently she is with Communication Security Group, Maritime Self-Defense Force, Japan

To clarify the relation of KSP to earlier works, we introduce the notation KSP $(\alpha n/r/m)$, where α is either one of C (meaning continuous), I (integer) or B (binary). This means that we have n variables of type α divided in r classes, with m constraints. Our problem is KSP $(Bn/r/1)$ in this notation. The special case of $r=n$ is well studied: Jacobsen [4] formulated KSP $(In/n/1)$ with a nonlinear objective function. Porteus and Yormark [9] gave a binary search algorithm to solve the same problem. Brown [1] first called this the *knapsack sharing problem*. We use the same terminology in a broader sense. Such a problem (with a linear or nonlinear objective function) is also referred to as the max-min or bottleneck allocation problem. See, e.g., Kaplan [5], Tang [10] and Luss [7] for this type of problems. Contrary, except for Kuno *et al.* [6] which dealt with KSP $(Cn/r/1)$, the general case of $r \neq n$ appears to have been rarely explored.

In this article we are primarily concerned with KSP $(Bn/r/1)$. Specifically, we give a fast algorithm to solve the relaxed problem KSP $(Cn/2/1)$, present a linear time heuristics for KSP $(Bn/r/1)$, and derive a pegging test to reduce the size of the problem.

2. AN UPPER BOUND TO KSP

We obtain an upper bound by solving the continuous relaxation of KSP.:

$$\begin{aligned} C(\text{KSP}) \quad & \text{maximize} \quad \min_{1 \leq k \leq r} \left\{ \sum_{j \in N_k} p_j x_j \right\} \\ & \text{subject to} \quad \sum_{j \in N} w_j x_j \leq \bar{c}, \\ & \quad \quad \quad 0 \leq x_j \leq 1, j \in N. \end{aligned}$$

Note that this is KSP $(Cn/r/1)$ to which a linear time solution algorithm is known [6]. We discuss the general case in 2.1, and in 2.2 a faster algorithm is given for the limited case of $r=2$.

2.1. Decomposition of $C(\text{KSP})$

First we introduce the auxiliary problem to solve $C(\text{KSP})$ through decomposition.

$$\begin{aligned} \text{Aux}_k(c^k): \quad & \text{maximize} \quad \sum_{j \in N_k} p_j x_j \\ & \text{subject to} \quad \sum_{j \in N} w_j x_j \leq c^k, \\ & \quad \quad \quad 0 \leq x_j \leq 1, j \in N_k, \end{aligned}$$

Note that this is the continuous KP which is easy to solve (Martello and Toth [8]). Let $\bar{x}^k(c^k)$ denotes a solution to $\text{Aux}_k(c^k)$ with the optimal objective value $\bar{z}^k(c^k)$. Here we introduce a slightly changed notation: we have $n_k := |N_k|$ items of class k , with the weight and value of the j th object in class k denoted as w_{kj} and p_{kj} respectively. We write x_{kj} correspondingly. By efficiency of an item we mean the value of that item per unit weight. This is denoted as $e_{kj} := p_{kj}/w_{kj}$. Items are assumed to be numbered in the non-increasing order of efficiency, i.e.,

$$1 \leq i < j \leq n_k \text{ implies } e_{ki} \geq e_{kj}. \quad (2)$$

Further, let w_j^k and p_j^k denote the cumulative weight and value up to the j th item in class k , i.e.,

$$w_j^k := \sum_{i=1}^j w_{ki}, p_j^k := \sum_{i=1}^j p_{ki}, j=0, \dots, n_k. \quad (3)$$

Then, $\bar{z}^k(\cdot)$ is the piecewise linear, concave and monotonically increasing function obtained by connecting $\{(w_j^k, p_j^k) \mid j=0, \dots, n_k\}$.

Now consider the problem;

$$\begin{aligned} \bar{C}(\text{KSP}): \quad & \text{maximize} \\ & \min_{1 \leq k \leq r} \{ \bar{z}^k(c^k) \} \\ & \text{subject to} \quad \sum_{1 \leq k \leq r} c^k \leq \bar{c}, \\ & \quad \quad \quad c^1, \dots, c^r \geq 0. \end{aligned}$$

Let $(\bar{c}^1, \dots, \bar{c}^r)$ denote an optimal solution to $\bar{C}(\text{KSP})$ with the corresponding objective value \bar{z} . Then $(\bar{x}^1(\bar{c}^1), \dots, \bar{x}^r(\bar{c}^r))$ is optimal to $C(\text{KSP})$ with \bar{z} as the corresponding objective value. Thus, \bar{z} gives an

upper bound to KSP. Note that \bar{z} is given as the solution to;

$$\sum_{k=1}^r \bar{c}^k(z) = \bar{c}, \tag{4}$$

where $\bar{c}^k(\cdot)$ denotes the inverse of $\bar{z}^k(\cdot)$. For

$$u_k := \min \{j \mid p_j^k \geq \bar{z}\} \tag{5}$$

the u_k th item of class k is said to be critical ($k=1, \dots, r$).

2.2. A fast algorithm to solve $C(KSP)$

In this part, we shall concentrate on the limited case of $r=2$. Then, functions $\bar{z}^k(\cdot)$ ($k=1,2$) can be shown in w - p plane as follows. In Fig. 1 let L_j and R_j stand for (w_j^1, p_j^1) and $(\bar{c} - w_j^2, p_j^2)$ respectively. Then the broken lines $L := L_0L_1 \dots L_{n_1}$ and $R := R_0R_1 \dots R_{n_2}$ represent, respectively, $\bar{z}^1(w^1)$ and $\bar{z}^2(\bar{c} - w^1)$. Thus, the solution to $\tilde{C}(KSP)$ is given at the unique intersection (\bar{c}^1, \bar{z}) of L and R , and \bar{z} gives an upper bound to KSP. (Without much loss of generality, we can assume that L and R actually meet.) To calculate \bar{z} we need to find the segments on which these two lines meet. By (5), these are the u_1 th and u_2 th segments respectively. To find u_1 and u_2 we first introduce the function;

$$\text{Interval}(R_j) := \min \{i \in N_1 \mid w_i^1 \geq \bar{c} - w_j^2\}. \tag{6}$$

Given R_j , $\text{Interval}(R_j)$ can be found in $O(\log n_1)$ steps by the standard binary search method. With this we again apply the binary search to obtain u_2 as follows;

Algorithm Right-Segment.

input: $n_1, n_2, (w_j^k), (p_j^k), \bar{c}$;

output: u_2 ;

begin

$left := n_2, right := 0$;

repeat

$mid := \lfloor (left + right) / 2 \rfloor$;

$i := \text{Interval}(R_{mid})$;

if R_{mid} lies above the segment $L_{i-1}L_i$ **then**

$left := mid$;

else $right := mid$;

until $left - right = 1$;

$u_2 := left$;

end.

In this algorithm the **repeat** loop is executed $O(\log n_2)$ times, and at each iteration we need $O(\log n_1)$ steps for a call to $\text{Interval}()$. Thus, the total complexity of Right-Segment is $O(\log n_1 \cdot \log n_2)$.

Next, we show how to find u_1 . Clearly, the line segment $R_{u_2-1}R_{u_2}$ crosses L within the interval $L_sL_{s+1} \dots L_r$, where $s := \text{Interval}(R_{u_2}) - 1$, and $t := \text{Interval}(R_{u_2-1})$. Thus, we have the following binary search algorithm to find u_1 .

Algorithm Left-Segment.

input: $n_1, n_2, (w_j^k), (p_j^k), \bar{c}, u_2$;

output: u_1 ;

begin

$left := \text{Interval}(R_{u_2}) - 1, right := \text{Interval}(R_{u_2-1})$;

repeat

$mid := \lfloor (left + right) / 2 \rfloor$;

if L_{mid} lies above the segment $R_{u_2-1}R_{u_2}$ **then**

$right := mid$;

else $left := mid$;

until $right - left = 1$;

$u_1 := right$;

end.

The complexity of Left-Segment is $O(\log n_1)$.

Now solving for (\bar{c}^l, \bar{z}) at which the two line segments $L_{u_1-1}L_{u_1}$ and $R_{u_2-1}R_{u_2}$ meet, we obtain

$$\bar{c}^l := (p_{u_2}^l - p_{u_1}^l + e_{1,u_1}w_{u_1}^l + e_{2,u_2}(\bar{c} - w_{u_2}^l)) / (e_{1,u_1} + e_{2,u_2}), \tag{7}$$

$$\bar{z} := p_{u_1}^l + e_{1,u_1}(\bar{c}^l - w_{u_1}^l). \tag{8}$$

Thus in total the upper bound \bar{z} can be found in $O(\log n_1 \cdot \log n_2)$ steps, which is faster than the linear-time algorithm of Kuno *et al.* [6].

3. A HEURISTIC ALGORITHM

Now we go back to the general case of $r (\geq 2)$ classes. Using u_1, \dots, u_r introduced in the previous section we define a feasible solution to KSP by $\hat{x}_{k,j} := 1$ if $j \leq u_k - 1$, and $\hat{x}_{k,j} := 0$ otherwise ($k=1, \dots, r$). The corresponding objective value is $\hat{z} := \min\{p_{u_1-1}^1, \dots, p_{u_r-1}^r\}$. This is referred to as the *trivial* solution to KSP. Starting with this we can further improve the solution successively by filling that part of the knapsack which is left vacant in the following way: at each step we compare the total value of items of each class already taken into the knapsack, and from the class of the smallest value we adopt the feasible item of highest efficiency. Let w^k and p^k denote the total weight and total value of the k th class items in the knapsack, and i_k stand for the item being considered for inclusion into the knapsack. Initially these are set in accordance with the trivial solution. The algorithm is described as follows.

Algorithm Greedy.

```

input:  $(n_j), (w_{k,j}), (p_{k,j}), \bar{c}, (u_j), \hat{x}$ 
output:  $\underline{x}, \bar{z}$ ;
begin
   $x := \hat{x}$ ; {comment: Initialize with the trivial solution}
  for  $k := 1$  to  $r$  do
     $w^k := w_{u_k-1}^k, p^k := p_{u_k-1}^k, i_k := u_k$ ;
  repeat {comment: Fill up the remaining vacancy}
     $l := \arg\{\min_{1 \leq k \leq r} p^k\}$ ;
    if  $i_l > n_l$  break;
    if  $w_{l,i_l} \leq \bar{c} - \sum_{k=1}^r w^k$  then
       $x_{l,i_l} := 1, p^l := p^l + p_{l,i_l}, w^l := w^l + w_{l,i_l}$ ;
    else  $x_{l,i_l} := 0$ ;
       $i_l := i_l + 1$ ;
  until  $i_k > n_k (k=1, \dots, r)$ ;
   $\bar{z} := \min\{p^1, \dots, p^r\}$ ;
end.
```

The solution resulting from this algorithm is referred to as the *greedy* solution. In the above algorithm this is denoted as $\underline{x} = (x_{k,j})$ with the objective value \bar{z} . The computational complexity of Greedy is $O(n)$, and the relative error of the greedy solution is at most;

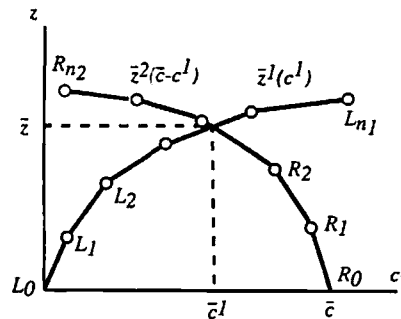


Fig. 1. Graphical representation of C(KSP).

$$\xi := (\bar{z} - \underline{z})/\underline{z} \times 100 (\%). \tag{9}$$

4. REDUCTION OF THE PROBLEM

Let $\gamma_k := 1/e_{k,u_k}$, and define for $j=1, \dots, n_k; k=1, \dots, r$,

$$\hat{z}(k;j) := \bar{z} - (p_{k,j}\gamma^k - w_{k,j})/\sum_{l=1}^k \gamma^l. \tag{10}$$

Then, analogous to the pegging test [3] for KP we have;

Theorem 1. *Let z denote the objective value of an arbitrary feasible solution to KSP. Then for the optimal solution $x^* = (x_{k,j}^*)$ we have;*

- (i) if $j < u_k$ and $z > \hat{z}(k;j)$ then $x_{k,j}^* = 1$.
- (ii) if $j > u_k$ and $z > \hat{z}(k;j)$ then $x_{k,j}^* = 0$.

Proof. Let KSP ($x_{k,j}=0$) be KSP with $x_{k,j}$ fixed at 0. Similarly, KSP ($x_{k,j}=1$) is KSP with $x_{k,j}=1$.

(i) Corresponding to KSP ($x_{k,j}=0$) with $j < u_k$, the j th segment of $\bar{z}^k(\cdot)$ is deleted and the broken line to its right is parallelly displaced by $(-w_{k,j}, -p_{k,j})$. The resulting function and its inverse are denoted as $\bar{z}^{k(j)}(\cdot)$ and $\bar{c}^{k(j)}(\cdot)$ respectively. Analogous to \bar{z} for KSP obtained by solving (4), an upper bound to KSP ($x_{k,j}=0$) is given as the solution to the equation;

$$\bar{c}^{k(j)}(z) + \sum_{l \neq k} \bar{c}^l(z) = \bar{c}. \tag{11}$$

This is denoted as $z = \bar{z}(k,j)$. Let $\hat{c}^l(\cdot)$ be the affine function obtained by extending the u_k th segment of $\bar{c}^l(\cdot)$. Similarly, $\hat{c}^{k(j)}(\cdot)$ denotes the affine function obtained from the (u_k-1) th segment of $\bar{c}^{k(j)}(\cdot)$. Note that the solution to

$$\hat{c}^{k(j)}(z) + \sum_{l \neq k} \hat{c}^l(z) = \bar{c} \tag{12}$$

is given by $z = \hat{z}(k,j)$. Further, from monotonicity and concavity of $\bar{z}^{k(j)}(\cdot)$ and $\bar{z}^l(\cdot)$ we have $\hat{z}(k,j) \geq \bar{z}(k,j)$. Thus, $\hat{z}(k,j)$ is also an upper bound to KSP ($x_{k,j}=0$). From this, (i) is immediate.

(ii) Similarly proved as (i). ■

5. A NUMERICAL EXAMPLE

Let $n_1 = n_2 = 9, \bar{c} = 175$ and the items in each class be as follows in Table 1. We start Right-Segment with *left* = 9 and *right* = 0, and get $u_1 = 5$. Next, we have *left* = 4 and *right* = 6 to start Left-Segment and obtain

Table 1. Data for numerical example

k	j	$w_{k,j}$	$p_{k,j}$	w_j^l	p_j^l	$e_{k,j}$
	1	15	45	15	45	3.0000
	2	10	25	25	70	2.5000
	3	19	40	44	110	2.1053
	4	31	27	75	137	0.8710
1	5	30	12	105	149	0.4000
	6	20	5	125	154	0.2500
	7	20	4	145	158	0.2000
	8	6	1	151	159	0.1667
	9	12	1	163	160	0.0833
	1	10	50	10	50	5.0000
	2	10	40	20	90	4.0000
	3	15	30	35	120	2.0000
	4	20	20	55	140	1.0000
2	5	30	15	85	155	0.5000
	6	24	7	109	162	0.2917
	7	5	1	114	163	0.2000
	8	45	8	159	171	0.1778
	9	8	1	167	172	0.1250

Table 2. Uncorrelated case

n	CPU time (s)			Total	Error(%)	Reduction(%)
	Setup	Greedy	Pegging			
50	0.0016	0.0006	0.0004	0.0025	6.297	53.60
100	0.0039	0.0002	0.0010	0.0050	2.780	53.95
200	0.0072	0.0008	0.0016	0.0096	0.837	70.55
500	0.0211	0.0014	0.0039	0.0264	0.403	65.04
1000	0.0437	0.0039	0.0080	0.0557	0.174	69.33
2000	0.0949	0.0086	0.0156	0.1191	0.069	75.06
5000	0.2658	0.0229	0.0437	0.3324	0.026	76.55
10 000	0.5910	0.0494	0.0980	0.7384	0.011	79.71
50 000	3.4906	0.2867	0.5037	4.2810	0.003	82.71
100 000	9.8441	0.8849	1.3308	12.0598	0.001	81.13
150 000	12.0287	1.1409	1.7673	14.9365	0.000	88.21
200 000	16.4103	1.5046	2.0803	19.9952	0.001	80.62
250 000	28.3986	3.2305	5.9580	37.5870	0.000	87.28

$u_1=5$. Thus, segments L_4L_5 and R_4R_5 meet, and we obtain $\bar{z}=148.33$. The trivial solution is $\hat{x}=(111100000111100000)$ with $\hat{z}=137$, and the greedy solution is $x=(1111100000111100101)$ with $\bar{z}=142$. Note that for $u_1=u_2=5$ we have $\gamma^1=0.4$ and $\gamma^2=0.5$. As for z in Theorem 1, let us take the greedy solution with $z=142$. Then, from Theorem 1 the following variables are fixed respectively at: $x_{1,1}^*=x_{1,2}^*=x_{1,3}^*=x_{1,4}^*=x_{2,1}^*=x_{2,2}^*=x_{2,3}^*=1, x_{2,8}^*=0$. Thus, out of 18 binary variables 8 (44.4%) are eliminated through this procedure.

6. COMPUTATIONAL EXPERIMENTS

Next, we show the result of a numerical experiment carried out under the following conditions: $n_1=n_2=n/2, \bar{c}=50n, (w_j)$ and (p_j) are mutually independent and uniformly random in $[1,1000]$.

Table 2 summarizes the results of this experiment, where for each n average of 20 independent runs are shown. The column *setup* refers to the CPU time (in seconds on a DECstation 3100) needed to generate data and sort them to the nonincreasing order of efficiency. *Greedy* and *pegging* shows the CPU time for computing the greedy solution and for carrying out the pegging tests respectively. *Error* is for the relative error evaluated by (9) and *reduction* means the percentage of the binary variables fixed either to 0 or 1 through the pegging procedure. From these, we observe the following:

70%–80% of the CPU time is taken for setting up the problem, especially for sorting the data.

The CPU time for greedy and pegging procedures is approximately proportional to n .

The error of the greedy solution decreases as n increases.

The pegging test eliminates 50%–90% of the binary variables if (w_j) and (p_j) are independently distributed.

Table 3 shows the result for the case where p_j is randomly distributed in $[w_j, w_j+200]$, while in Table 4 p_j is given as w_j+100 . These represent weak and strong correlations between (w_j) and (p_j) . We conclude from these tables:

CPU time and accuracy remain almost the same for correlated and uncorrelated cases.

The effect of pegging test sharply deteriorates with the degree of correlation between weight and value of items.

Table 3. Weakly correlated case

n	CPU time (s)			Total	Error(%)	Reduction(%)
	Setup	Greedy	Pegging			
50	0.00	0.00	0.00	0.00	10.669	10.30
100	0.00	0.00	0.00	0.01	3.909	11.50
200	0.01	0.00	0.00	0.01	1.289	24.22
500	0.02	0.00	0.00	0.03	0.446	30.13
1000	0.04	0.00	0.01	0.06	0.286	30.07
2000	0.10	0.01	0.02	0.12	0.102	38.12
5000	0.27	0.02	0.04	0.34	0.033	45.20
10,000	0.59	0.05	0.09	0.74	0.016	52.91
50,000	3.49	0.29	0.49	4.26	0.004	42.16
100,000	7.42	0.68	0.99	9.10	0.002	53.10
150,000	15.89	1.47	2.09	19.45	0.002	46.60
200,000	21.83	2.04	2.90	26.77	0.001	57.96
250,000	23.23	2.49	4.48	30.21	0.001	54.79

Table 4. Strongly correlated case

n	CPU time (s)			Error(%)	Reduction(%)	
	Setup	Greedy	Pegging			
50	0.00	0.00	0.00	0.00	11.451	2.40
100	0.00	0.00	0.00	0.00	6.180	3.90
200	0.01	0.00	0.00	0.01	2.187	10.65
500	0.02	0.00	0.00	0.03	1.240	1.32
1000	0.04	0.00	0.01	0.05	0.591	0.30
2000	0.09	0.01	0.01	0.11	0.284	6.31
5000	0.25	0.03	0.04	0.32	0.114	2.30
10,000	0.55	0.05	0.09	0.70	0.052	2.11
50,000	3.45	0.33	0.47	4.26	0.008	8.81
100,000	7.33	0.77	0.94	9.05	0.005	2.82
150,000	15.42	1.61	1.89	18.92	0.004	1.15
200,000	21.75	2.30	2.60	26.65	0.003	2.87
250,000	22.76	2.41	3.98	29.15	0.002	0.01

7. CONCLUSION

We have formulated KSP, presented heuristic and reduction algorithms for this problem, and analysed the behavior of the developed algorithms through computational experiments. Except for the time required to set up problems, with the developed algorithm we were able to solve (to less than 0.005% errors) KSPs with 250,000 items in less than 10 s in our computing environment. However, exact algorithms to solve KSP is still under development and will be reported in due course.

REFERENCES

1. Brown, J. R., The knapsack sharing problem. *Operations Research*, 1979, 27, 341–355.
2. Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, 1978.
3. Inargiola, G. P. and Korsh, J. F., Reduction algorithms for zero-one single knapsack problems. *Management Science*, 1973, 20, 460–463.
4. Jacobsen, S., On marginal allocation in single constraint min-max problems. *Management Science*, 1971, 17, 780–783.
5. Kaplan, S., Application of programs with maximum objective functions to problems of optimal resource allocation. *Operations Research*, 1974, 22, 802–807.
6. Kuno, T., Konno, H. and Zemel, E., A linear-time algorithm for solving continuous maximin knapsack problems. *Operations Research Letters*, 1991, 10, 23–26.
7. Luss, H., Minimax resource allocation problems: optimization and parametric analysis. *European Journal of Operational Research*, 1992, 60, 76–86.
8. Martello, S. and Toth, P., *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, New York, 1990.
9. Porteus, E. L. and Yormark, J. S., More on min-max allocation. *Management Science*, 1972, 17, 502–507.
10. Tang, C. S., A max-min allocation problem: its solutions and applications. *Operations Research*, 1988, 36, 359–367.