A note on the max-min 0–1 knapsack problem

HIROSHI IIDA

oggi@res.otaru-uc.ac.jp

*Dept. of Information and Management Science, Otaru University of Commerce, 5-21 Midori
 3-chome Otaru Hokkaido, 047-8501 Japan*

Received April 28, 1998; Revised November 6, 1998

Editor:

Abstract. In this paper we propose new lower and upper bounds for the max-min 0–1 knapsack problem, employing a mixture of two relaxations. In addition, in order to expose whether the bounds are practical or not, we implement a method incorporating the bounds to achieve an optimal solution of the problem.

Keywords: knapsack problem, surrogate relaxation, linear programming relaxation, branch-and-bound

1. Introduction

The max-min 0–1 knapsack (MNK) problem was recently introduced by Yu [3] with several practical applications. While the value (profit) of each item in the classical 0–1 knapsack problem (KP) is constant, we consider that the values vary according to a *scenario*. The MNK is to select items so that the minimum total value of the selected items gained under all scenarios is maximum under the constraint that the total weight of the selected items does not exceed a given capacity. Here we formulate the MNK problem as follows:

$$z_{\text{MNK}} = \max_x \min_{s \in S} \left\{ \sum_{i=1}^n v_i^s x_i \mid \sum_{i=1}^n a_i x_i \leq b; x_i \in \{0, 1\}, i \in N \right\}, \quad (1)$$

where $N := \{1, 2, \dots, n\}$, and S is a set of scenarios in which each item $i \in N$ has value v_i^s under scenario $s \in S$. In addition, each item i has weight a_i , satisfying $a_i \leq b$ and $\sum_{i \in N} a_i > b$. Without loss of generality we will assume that all v_i^s and a_i are positive. A solution $I(\subseteq N)$ is said to be *feasible*, if it satisfies $\sum_{i \in I} a_i \leq b$. Also we naturally identify an n -vector of 0–1 variables $x = (x_i)_{i \in N}$ with a solution I as $x_i = 1 \Leftrightarrow i \in I$.

The MNK reduces to KP in the case where the number of scenarios, which we denote as $|S|$, is one. The MNK thus includes KP as a special case, and is $\mathcal{NP}$ -hard. In addition, MNK is strongly $\mathcal{NP}$ -hard when the number of scenarios is unbounded, which has also been proved in [3].

As presented in [3], a surrogate relaxation for MNK can be prescribed as a KP. We here cite the relaxation:

$$\begin{aligned} z_U(\mu) = \max_x \sum_{i=1}^n v_i(\mu)x_i, \quad v_i(\mu) = \sum_{s \in S} \mu_s v_i^s, \quad \text{where } \mu_s \geq 0 \ (s \in S), \quad \sum_{s \in S} \mu_s = 1 \\ \text{subject to } \sum_{i=1}^n a_i x_i \leq b, \quad x_i \in \{0, 1\}, \quad i \in N. \end{aligned} \quad (2)$$

In [3], a subgradient procedure is used in obtaining tight lower and upper bounds for MNK. In the procedure, in every iteration, the KP (2) with a multiplier vector $(\mu_s)_{s \in S}$ is exactly solved, and the result is exploited to improve the multiplier vector in order to have a stronger relaxation. However, as it is well-known, the tradeoff between the sharpness of a bound and the computing time to obtain the bound on exploiting the branch-and-bound is substantial. Therefore, in this paper an attempt is made to apply linear programming relaxation (LP relaxation, for short) to the KP (2), and quickly obtain an alternative upper bound well-known as the Dantzig upper bound (Dantzig [1]). In addition, we devise a simple procedure to improve the bound.

The remainder of this paper is organized as follows: We present new lower and upper bounds for MNK in Section 2, and illustrate a method incorporating the bounds in Section 3. Computational experiments are presented in Section 4.

2. Lower and upper bounds

First of all we should determine a multiplier vector $(\mu_s)_{s \in S}$ before applying LP relaxation to the KP (2). Here we will introduce three types of multiplier vectors.

1. The one inversely proportional to the total value of scenario s :

$$\mu_s = \frac{1}{\sum_{i=1}^n v_i^s}, \quad \text{for each } s \in S. \quad (3)$$

Preliminary computational experiments showed that this multiplier is better than both a uniform one $\mu_s = 1$ and a proportional one $\mu_s = \sum_{i=1}^n v_i^s$. We remark that we should determine a multiplier vector so that the KP (2) has an optimal value as small as possible. In fact, among the previous three types, the proportional one is worst. This implies that, on a multiplier corresponding to scenario s , the larger $\sum_{i=1}^n v_i^s$ is, the smaller μ_s should be. The remaining two also follow this argument.

2. The one inversely proportional to the total efficiency of scenario s :

$$\mu_s = \frac{1}{\sum_{i=1}^n v_i^s / a_i}, \quad \text{for each } s \in S. \quad (4)$$

3. Before describing the third, we introduce some notation: With $z_U(\mu_s)$ we denote a problem prescribed by assuming that there exists only one scenario s on given

MNK, which is formulated as follows:

$$z_U(\mu_s) = \max_x \left\{ \sum_{i=1}^n v_i^s x_i \mid \sum_{i=1}^n a_i x_i \leq b; x_i \in \{0, 1\}, i \in N \right\}. \quad (5)$$

$\bar{z}_U(\mu_s)$ denotes a Dantzig upper bound obtained by applying LP relaxation to the KP (5). Then we employ the inverse of $\bar{z}_U(\mu_s)$ as the third multiplier corresponding to scenario s .

On the subgradient procedure in [3], the lower bound will be improved by exploiting the solution which gives an optimal value of the KP (2) in every iteration. In a similar way we will introduce a simple procedure which not only improves our upper bound, but also produces a lower bound:

```

1. IterProc(In:  $(\mu_s)_{s \in S}$ , Out: LB, UB)
2.   Normalize received  $(\mu_s)_{s \in S}$ ;                                /* three types of  $(\mu_s)_{s \in S}$  */
3.   Apply LP relaxation to the KP (2) with  $(\mu_s)_{s \in S}$ ,
4.   and obtain UB and the solution  $x^*$  which gives UB;
5.    $LB := \min_{s \in S} \sum_{i \in N} v_i^s x_i^*$ ;
6.   while (1) {
7.     foreach  $s \in S$   $\{\mu_s := (\mu_s)^2\}$                                 /* bias */
8.     Normalize  $(\mu_s)_{s \in S}$ ;
9.     Apply LP relaxation to the KP (2) with  $(\mu_s)_{s \in S}$ ,
10.    and obtain tmpUB and the solution  $x^*$  which gives tmpUB;
11.     $tmpLB := \min_{s \in S} \sum_{i \in N} v_i^s x_i^*$ ;
12.    if (tmpLB > LB)
13.       $LB := tmpLB$ ;
14.    if (tmpUB < UB)
15.       $UB := tmpUB$ ;
16.    else
17.      break;                                /* exit infinite loop and return */
18.  }                                           /* infinite loop */

```

At the beginning in the iteration, each multiplier is multiplied by itself. This operation gives a bias to each multiplier so that the difference between any pair of multipliers will be gradually widened. After the operation, the multiplier vector is normalized so that $\sum_{s \in S} \mu_s = 1$. The iteration continues until UB does not improve. On the solution x^* note that the last taken item is excluded after computing the upper bound, if $b < \sum_{i \in N} a_i x_i^*$ holds. This is to ensure the feasibility of the solution which gives LB.

We remark that a multiplier vector that $\mu_s = 1$ and other $|S|-1$'s are zero satisfies the condition on the one in the KP (2). Therefore for any $s \in S$, the problem (5) is a relaxation for MNK. Then we employ another candidate of $\min_{s \in S} \bar{z}_U(\mu_s)$ for an upper bound. Consequently we exploit four candidates for an upper bound. In

the next section we illustrate a method incorporating the bounds introduced in this section.

3. A method for the MNK problem

We have implemented our method based on the branch-and-bound and best-first search schemes according to [3]. In this section with N_0 we will denote a subset of the set N in which $x_i = 0$, while with N_1 a subset of N in which $x_i = 1$. Also $N_2 := N \setminus (N_0 \cup N_1)$.

On each node of the binary search-tree, we consider a subproblem of given MNK prescribed by the triplet $\{N_0, N_1, N_2\}$ which is corresponding to the node. Also the surrogate relaxation of the subproblem is formulated as follows:

$$z'_U(\mu) = \max_{I \subset N_2} \left\{ \sum_{i \in I \cup N_1} v_i(\mu) \mid \sum_{i \in I \cup N_1} a_i \leq b, v_i(\mu) = \sum_{s \in S} \mu_s v_i^s \right\}, \quad (6)$$

where $\mu_s \geq 0$ for any $s \in S$, and $\sum_{s \in S} \mu_s = 1$. On applying the procedure *IterProc* to a node three times with different types of multiplier vectors separately, we take N_0 into account: On the formulae (3) and (4) we compute each multiplier by the value or efficiency of items only in $N_1 \cup N_2$ respectively; The third one is computed so that only the item in N_2 is considered whether we take it or not. The modification of *IterProc* to be applicable to (6), which is only on the part of applying LP relaxation to (1), is straightforward.

On the best-first search, active nodes (subproblems) are stored in a list in a nonincreasing order of the upper bound computed on each. Also the lower bound obtained simultaneously will be exploited to improve an incumbent value. Hereafter we will denote the incumbent value with z^* . The tree search is performed as follows: After taking a node from the head of the list, if the upper bound of the node is less than or equal to z^* then the processing to optimality will be terminated immediately; otherwise, on the node,

1. If the total weight of N_2 is suitable to the residual capacity $b - \sum_{i \in N_1} a_i$, we discard the node after computing $\min_{s \in S} \sum_{i \in N_1 \cup N_2} v_i^s$ in order to improve z^* if possible.
2. If in N_2 there exists no item suitable to the residual capacity, we discard the node after computing $\min_{s \in S} \sum_{i \in N_1} v_i^s$.

After the two tests, in N_2 , any item unsuitable to the residual capacity is moved into N_0 . Finally the node is expanded according to a branching variable determined as item $i \in N_2$ giving a maximum of $\sum_{s \in S} v_i^s$.

4. Computational experiments

Before presenting computational experiments, we illustrate the instances of MNK provided for the experiments. They have been taken from Tables I, II and III in [3], where

- a_i : uniformly distributed in $(0, 100)$
- $\hat{v}_i$: a base value for v_i^s , uniformly distributed in $(0, 100)$
- v_i^s : uniformly distributed in $(\hat{v}_i(1 - \delta), \hat{v}_i(1 + \delta))$ for all $s \in S$
- b : $\sum_{i=1}^n a_i/m$.

We provide all these data as integer type. While computational experiments are performed on three values for n as 20, 40 and 60 in [3], we deal with only 60. It should be pointed out that the larger the δ is, the harder the instance of MNK is, which is mentioned in [3].

Now we present computational experiments in Table 1. Our method has been implemented in C, and the reference machine is SPARCstation-20. In Table 1, each line expresses the average of 100 instances. Computing times are shown in the columns for CPU, expressed in seconds. Columns for BBN indicate the average total number of visited nodes, including those which are not expanded. Columns for z_{XL} and z_{XU} express the average of our lower and upper bounds on the root node respectively. The Δ_1 expresses the gap between the bounds. On the other hand the columns for Yu are as is in [3] for reference, where the reference machine is IBM3090, and the algorithm was implemented in Pascal. Also the Δ_2 expresses the gap between z_L lower and z_U upper bounds in [3]. As we can see in Table 1, Δ_1 is roughly three or four times worse than Δ_2 .

To the best of our knowledge, the speed comparison between Mips and SPECint92 is impossible. Still, roughly speaking, our reference machine would not be ten times faster than the one in [3]¹. It could thus be concluded that our method is practical to solve MNK, which is remarkable in the case where $\delta = 0.3$, that is, values are distributed in a rather short range. On the other hand our method shows poor performance as $|S|$ becomes large. As the $|S|$ grows, for instance, it will be expensive to compute the three types of initial multiplier vectors. Here is an additional remark that, on our method, BBN tends to be best in the case where $m = 2$ on any pair of δ and $|S|$. It is a strange behavior, since the total number of feasible solutions is largest in the case where $m = 2$ among $m = 2, 3$ and 4 in general. Therefore BBN (and CPU) should be smallest in the case where $m = 4$ as the method proposed in [3] behaves so.

We would like to add a concluding remark: the computational experiments exhibit that the proposed bounds are sufficient to solve the max-min 0–1 knapsack problem in the case where both the number of items and scenarios are moderate, which is due to the lightness of the processing in contrast to the one by Yu. In other words, in such a case, our bounds are enough to solve the problem in reasonable computing time, although they are not so tight compared with those by Yu.

Acknowledgments

The author thanks an anonymous referee for valuable comments. Also the numerical performance data of IBM3090/200E is a courtesy of Prof. Gang Yu and the computation center at the University of Texas at Austin.

Table 1. The comparison between the two methods ($n = 60$)

δ	$ S $	m	by Yu*		BBN	CPU (sec.)	by Iida**		Δ_1	Δ_1/Δ_2
			BBN	CPU (sec.)			z_{XL}	z_{XU}		
0.3	10	2	43.0	227.5	190.78	3.0	2315.68	2334.50	0.0081	1.92
		3	36.4	200.8	276.12	3.6	1888.64	1914.17	0.0133	5.11
		4	29.6	120.1	231.44	3.0	1607.32	1638.67	0.0191	4.06
	20	2	42.4	217.9	256.62	6.0	2279.71	2299.90	0.0088	2.51
		3	35.4	234.3	321.24	7.3	1860.04	1887.90	0.0148	2.46
		4	29.8	200.7	295.82	6.3	1587.44	1617.49	0.0186	4.53
	30	2	40.8	221.5	255.00	8.1	2281.29	2302.36	0.0092	1.87
		3	36.0	229.2	320.94	9.6	1823.97	1851.94	0.0151	2.55
		4	29.8	190.2	274.04	7.9	1550.45	1581.96	0.0199	2.31
0.6	10	2	43.7	231.3	341.70	4.9	2219.32	2251.32	0.0142	2.15
		3	36.2	270.8	494.18	6.8	1803.75	1850.61	0.0253	2.38
		4	30.0	204.9	442.96	5.7	1531.41	1582.46	0.0323	7.34
	20	2	42.2	245.5	406.60	9.3	2149.33	2186.63	0.0171	1.87
		3	35.0	243.3	626.38	13.2	1742.04	1793.76	0.0288	3.03
		4	33.0	212.1	428.56	8.6	1499.29	1556.35	0.0367	3.52
	30	2	43.2	241.3	465.30	13.8	2163.33	2200.26	0.0168	3.81
		3	35.8	227.6	696.72	20.6	1748.49	1807.73	0.0328	2.32
		4	31.6	191.2	577.28	15.8	1497.23	1558.01	0.0390	2.07
0.9	10	2	44.8	334.0	726.48	9.1	2122.82	2179.98	0.0262	2.31
		3	38.0	322.4	882.92	10.7	1743.43	1823.58	0.0440	2.36
		4	31.0	253.2	1099.22	12.4	1514.28	1607.36	0.0579	3.46
	20	2	44.2	262.4	965.00	20.0	2104.26	2179.34	0.0345	3.28
		3	36.4	246.9	1338.56	26.0	1664.19	1764.02	0.0566	2.94
		4	31.4	207.4	1187.24	21.4	1412.45	1510.45	0.0649	2.43
	30	2	44.8	252.3	1290.10	36.9	2013.84	2091.58	0.0372	2.02
		3	38.0	234.9	1829.58	48.0	1652.59	1754.82	0.0583	3.49
		4	30.0	189.1	1557.82	38.9	1389.79	1506.62	0.0775	3.74

* : solved on IBM3090/200E (2CPUs, rated 32Mips)

** : solved on SPARCstation-20 model 71 (125.8SPECint92/121.2SPECfp92)

 z_{XL} : the average lower bound on the root node z_{XU} : the average upper bound on the root node Δ_1 : $(z_{XU} - z_{XL})/z_{XU}$ Δ_2 : $(z_U - z_L)/z_U$, as is in [3]

Notes

1. For instance, SONY NWS-7900 of 203Mips is 117SPECint92/125SPECfp92, and HP 9000/700 model 755/125 of 154Mips is 132.8SPECint92/195.7SPECfp92 in a press release respectively.

References

1. Dantzig, G.B., “Discrete variable extremum problems,” *Operations Res.* **5**, 266–277 (1957).
2. Iida, H., *On solving the max-min 0-1 knapsack problem*, Research Report: IS-RR-97-0025F, JAIST Hokuriku, 923-1292 Japan, 1997.
3. Yu, G., “On the max-min 0-1 knapsack problem with robust optimization applications,” *Operations Res.* **44**(2) 407–415 (1996).