Available online at www.sciencedirect.com

ScienceDirect

Computers & Operations Research 35 (2008) 2034–2048

computers & operations research

www.elsevier.com/locate/cor

Heuristic and exact algorithms for the max–min optimization of the multi-scenario knapsack problem

Fumiaki Taniguchi, Takeo Yamada*, Seiji Kataoka

Department of Computer Science, The National Defense Academy, Yokosuka, Kanagawa 239-8686, Japan

Available online 13 November 2006

Abstract

We are concerned with a variation of the standard 0–1 knapsack problem, where the values of items differ under possible S scenarios. By applying the 'pegging test' the ordinary knapsack problem can be reduced, often significantly, in size; but this is not directly applicable to our problem. We introduce a kind of surrogate relaxation to derive upper and lower bounds quickly, and show that, with this preprocessing, the similar pegging test can be applied to our problem. The reduced problem can be solved to optimality by the branch-and-bound algorithm. Here, we make use of the surrogate variables to evaluate the upper bound at each branch-and-bound node very quickly by solving a continuous knapsack problem. Through numerical experiments we show that the developed method finds upper and lower bounds of very high accuracy in a few seconds, and solves larger instances to optimality faster than the previously published algorithms.

Keywords: Knapsack problem; Max–min combinatorial optimization; Robust optimization

1. Introduction

Knapsack problem [1,2] has been studied extensively in operations research and computer science as a fundamental combinatorial optimization problem. In this article, we are concerned with a variation of this problem, where the *profits* of items differ under possible S *scenarios*. By p_j^s we denote the value of item j under scenario s $(s = 1, 2, \ldots, S)$, and x_j is the decision variable that takes value 1 if item j is adopted and 0 otherwise $(j = 1, 2, \ldots, n)$. Then,

$$z^s(x) := \sum_{j=1}^{n} p_j^s x_j \tag{1}$$

is the total value of the solution $x = (x_j)$ under scenario s, and thus we have S objective functions to maximize. Contrary to the profits, the *weight* of item j is assumed to be constant w_j through all scenarios, and the knapsack *capacity* is c.

Such a multi-objective optimization problem has been investigated under the framework of *multi-criteria* decision making [3,4]. An approach to this problem is *max–min* optimization [5], where we maximize the *minimum*

* Corresponding author. Tel.: +81 46 841 3810; fax: +81 46 844 5911.
E-mail address: yamada@nda.ac.jp (T. Yamada).

of $z^s(x),\ s = 1, 2, \ldots, S$. In some literature this is referred to as the *robust optimization* [6]. Thus, we formulate the *max–min* 0–1 *knapsack problem* as

MKP:

$$\text{maximize} \quad z(x) := \min_{1 \leqslant s \leqslant S} \left\{ \sum_{j=1}^{n} p_j^s x_j \right\} \tag{2}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_j \leqslant c, \tag{3}$$

$$x_j \in \{0, 1\}, \quad j = 1, 2, \ldots, n. \tag{4}$$

Without much loss of generality, throughout the paper we assume that

A$_1$. $p_j^s\ (j = 1, 2, \ldots, n; s = 1, 2, \ldots, S)$ are non-negative integers.
A$_2$. $w_j\ (j = 1, 2, \ldots, n)$ and c are positive integers.
A$_3$. $\sum_{j=1}^{n} w_j > c$ and $w_j < c\ (j = 1, 2, \ldots, n)$.

Such a problem arises naturally when uncertainty is unnegligible. As an example, we may have n possible projects to invest within a fixed budget c. Project j costs w_j, while the profit may be either p_j^1, p_j^2 or p_j^3 depending on the economic condition s (=1: good, 2: fair or 3: poor) of the next few years. If we wish to maximize the worst-case profit, we naturally have MKP.

MNK is $\mathcal{NP}$-hard [7], since for $S = 1$ the problem is the standard 0–1 knapsack problem which is already $\mathcal{NP}$-hard [1]. We may solve small instances of MKP using free or commercial IP solvers [8]. Yu [6], Kouvelis–Yu [9] and Iida [10] gave branch-and-bound algorithms for MKP, and solved problems with up to $n \leqslant 90$ items and $S \leqslant 30$ scenarios. In this paper, we present an algorithm that combines (i) surrogate relaxation to find upper and lower bounds quickly, (ii) a new pegging test for MKP and (iii) a surrogate-based branch-and-bound method to solve the reduced problem. Combining these, we are often able to solve MKPs with up to $n = 1000$ items to optimality faster than the previously published algorithms.

In Section 2, we introduce the surrogate relaxation to derive upper and lower bounds quickly. Making use of this relaxation, the pegging test is extended to MKP in Section 3. Then, the reduced problem is solved to optimality by the 'surrogate-based' branch-and-bound algorithm developed in Section 4. Through a series of numerical tests, in Section 5 we evaluate the performance of the developed methods, and Section 6 gives conclusion.

2. Upper and lower bounds

We follow Yu [6] to introduce the *surrogate relaxation* [11] and derive an upper bound. In addition, we further relax the 0–1 constraint (4) continuously. The result is a continuous knapsack problem which is easily solved [1], and from this solution we readily obtain a feasible solution, and thus a lower bound to MKP.

2.1. Surrogate relaxation

Let $\lambda = (\lambda_s)$ be an arbitrary vector in $\varDelta := \{(\lambda_1, \lambda_2, \ldots, \lambda_S) | \sum_{s=1}^{S} \lambda_s = 1,\ \lambda_s \geqslant 0,\ \forall s\}$. Then, from

$$\min_{1 \leqslant s \leqslant S} \left\{ \sum_{j=1}^{n} p_j^s x_j \right\} \leqslant \sum_{s=1}^{S} \lambda_s \sum_{j=1}^{n} p_j^s x_j$$

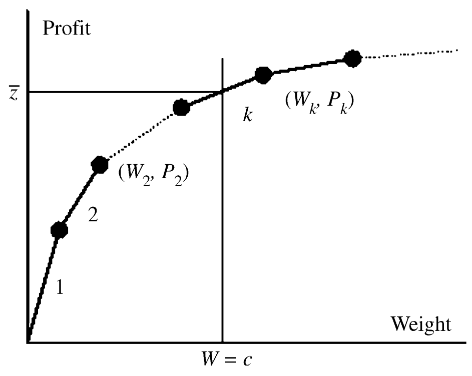

Fig. 1. Solution of SMKP(λ).

we obtain the following *surrogate relaxation problem* [9]:

SMKP(λ):

$$\text{maximize} \quad \sum_{j=1}^{n} \bar{p}_j(\lambda) x_j \tag{5}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_j \leqslant c, \tag{6}$$

$$0 \leqslant x_j \leqslant 1, \quad j = 1, 2, \ldots, n, \tag{7}$$

where

$$\bar{p}_j(\lambda) := \sum_{s=1}^{S} \lambda_s p_j^s. \tag{8}$$

We note that x_j is also relaxed continuously as (7), and thus for a fixed $\lambda \in \Delta$, SMKP(λ) is a continuous knapsack problem. In what follows we denote $\bar{p}_j := \bar{p}_j(\lambda)$ if this is not confusing, and assume

B$_1$. Items are numbered in the non-increasing order of $r_j := \bar{p}_j / w_j$.

By $\bar{x}(\lambda)$ we denote an optimal solution to SMKP(λ) with the corresponding optimal objective value $\bar{z}(\lambda)$.

Let W_j and P_j be, respectively, the *accumulated* weight and profit, i.e.,

$$W_j := \sum_{i=1}^{j} w_i, \quad P_j := \sum_{i=1}^{j} \bar{p}_i, \quad j = 0, 1, \ldots, n.$$

Item k is said to be *critical* if it satisfies $W_{k-1} \leqslant c < W_k$. Then, the broken line connecting $(W_0, P_0), (W_1, P_1), \ldots, (W_n, P_n)$ and (∞, P_n) gives a piecewise-linear, monotonically non-decreasing, concave function (see Fig. 1), and the intersection of this with the vertical line $W = c$ gives the upper bound $\bar{z}(\lambda)$.

Viewed as a function of λ, the following properties of $\bar{z}(\lambda)$ can be easily shown, analogous to the case of *Lagrangian relaxation* [12,13]. That is, (i) $\bar{z}(\lambda)$ is a piecewise-linear, convex function of λ and (ii) if $\bar{z}(\lambda)$ is differentiable at λ,

$$\partial \bar{z}(\lambda)/\partial \lambda_s = z^s(\bar{x}(\lambda)) \equiv \sum_{j=1}^{n} p_j^s \bar{x}_j(\lambda), \quad s = 1, 2, \ldots, S. \tag{9}$$

2.2. Subgradient method

To find an upper bound with $\bar{z}(\lambda)$ as small as possible, we solve the following *surrogate dual problem* [11]:

$$\text{minimize } \bar{z}(\lambda) \quad \text{subject to } \lambda \in \Delta.$$

To solve this, we employ the subgradient method, where subgradient is the vector $g := \partial\bar{z}(\lambda)/\partial\lambda$ whose element is given by (9). The *search direction* d is determined as the projection of $-g$ onto Δ, i.e.,

$$d := (-g + \bar{g}\mathbf{1})/\| - g + \bar{g}\mathbf{1}\|,$$

where $\bar{g} := \sum_{s=1}^{S} g_s/S$, and $\mathbf{1}$ is the vector with all components being 1. This is indeed a non-increasing direction, as we have

$$\partial\bar{z}(\lambda + \alpha d)/\partial\alpha|_{\alpha=0} = g \cdot d \leqslant 0.$$

Using this direction d, we can construct a *subgradient algorithm* [12,14], and by $\lambda^{\dagger}$ we denote the (sub)optimal solution to the surrogate dual problem obtained from this algorithm. In the case of two scenarios ($S = 2$), instead of the subgradient algorithm we may employ the *binary search* method to obtain $\lambda^{\dagger}$ more efficiently. Thus, we obtain an upper bound to MKP as $\bar{z} := \bar{z}(\lambda^{\dagger})$.

2.3. Lower bounds

For an arbitrary $\lambda \in \Delta$, $\bar{x}(\lambda)$ satisfies (3). If this also satisfies the 0–1 constraint (4), this is feasible to MKP; hence, the corresponding objective value gives a lower bound to the original problem. On the other hand, if some components of $\bar{x}(\lambda)$ violate (4), we still obtain a feasible solution by replacing all those fractional components with 0. The solution thus obtained may further be improved by some simple *greedy* procedure. In the subgradient procedure, each time we solve SMKP(λ) we thus get a lower bound, and the largest one found in the whole process is henceforth denoted as $\underline{z}$.

3. Pegging test

Pegging test [15–17] is well known for the ordinary 0–1 knapsack problem, where by applying this test many variables are fixed either at 0 or 1, and removing these we are left with a problem of (often significantly) reduced size. Although this technique is unsuitable to MKP in its original form, we can make this applicable to our problem by preceding it with the surrogate relaxation of the previous section.

Assume that we have the suboptimal surrogate multiplier $\lambda^{\dagger}$, the corresponding upper bound $\bar{z} = \bar{z}(\lambda^{\dagger})$ and a lower bound $\underline{z}$ to MKP, and let us consider SMKP($\lambda^{\dagger}$). For an arbitrary $j = 1, 2, \ldots, n$, we write $\bar{p}_j := \bar{p}_j(\lambda^{\dagger})$, and $z^{\star}_{j,\delta}$ denotes the optimal objective value to MKP with an additional constraint $x_j = \delta$, where δ is either 0 or 1. Similarly, $\bar{z}_{j,\delta}$ is the optimal objective value to SMKP($\lambda^{\dagger}$) with $x_j = \delta$. These can be written as

$$z^{\star}_{j,\delta} := \max\{(2)|(3), (4),\ x_j = \delta\},$$
$$\bar{z}_{j,\delta} := \max\{(5)|(6), (7),\ x_j = \delta\}.$$

If the optimal objective value to MKP is $z^{\star}$, for $j = 1, 2, \ldots, n$, we have the following:

$$z^{\star} = \max\{z^{\star}_{j,0}, z^{\star}_{j,1}\},$$

$$z^{\star}_{j,\delta} \leqslant \bar{z}_{j,\delta}.$$

Then, if

$$\bar{z}_{j,0} < \underline{z}, \tag{10}$$

it is not possible that $x^{\star}_j = 0$ in any optimal solution $x^{\star} = (x^{\star}_j)$ to MKP, i.e., we necessarily have $x^{\star}_j = 1$.

To determine (10) quickly, the following shortcut is usually taken. Here, we assume $\mathbf{B}_1$ again. Then, if for any $u < k$ we set $x_j = 0$, it is known [15–17] that

$$\bar{z}_{j,0} \leqslant \bar{z} - \theta_j, \tag{11}$$

where we define the *threshold* for item *j* by

$$\theta_j := \bar{p}_j - r_k w_j. \tag{12}$$

Then, if

$$\bar{z} - \underline{z} < \theta_j$$

from (11) we have $\bar{z}_{j,0} < \underline{z}$, and thus $x_j^\star = 1$. Taking also the case of $\bar{z}_{j,1} < \underline{z}$ into account, we have the following.

Theorem 1. *For any optimal solution* $x^\star = (x_j^\star)$ *to* MKP, *both of the following hold*:

(i) $\bar{z} - \underline{z} < \theta_j \Rightarrow x_j^\star = 1$,
(ii) $\bar{z} - \underline{z} < -\theta_j \Rightarrow x_j^\star = 0$.

The ordinary pegging test was applicable only to the standard 0–1 knapsack problems [15–17]. By preprocessing MKP with surrogate relaxation, the pegging test has been extended to MKP through the above theorem. Thus, by applying this theorem some variables are fixed, and removing these we obtain an MKP of (often substantially) reduced size. We call this *reduction* by pegging test, and thus obtain a reduced MKP.

4. Surrogate-based branch-and-bound

Let F_0 (F_1, resp.) be the set of variables fixed at 0 (1, resp.) by the pegging test of Section 3. In the branch-and-bound methods to be stated below, we define U as the set of unfixed variables, and $\hat{w}$ and $\hat{p}^s$ denote the sum of the weights and profits of the items fixed at 1. Initially these are

$$U := \{1 \leqslant j \leqslant n \mid j \notin F_0 \cup F_1\}, \tag{13}$$

$$\hat{w} := \sum_{j\in F_1} w_j, \quad \hat{p}^s := \sum_{j\in F_1} p_j^s \quad (s = 1, 2, \ldots, S). \tag{14}$$

We introduce the *subproblem* of MKP as
P$(U, \hat{w}, (\hat{p}^s))$:

$$\begin{aligned} \text{maximize} \quad & \min_{1\leqslant s\leqslant S} \left\{ \sum_{j\in U} p_j^s x_j + \hat{p}^s \right\} \\ \text{subject to} \quad & \sum_{j\in U} w_j x_j \leqslant c - \hat{w}, \\ & x_j \in \{0, 1\}, \quad \forall j \in U. \end{aligned}$$

By $z^\star(U, \hat{w}, (\hat{p}^s))$ we denote the optimal objective value to this problem. Clearly, by solving this with parameters given in (13) and (14), the original MKP is solved.

Next, using $\lambda^\dagger$ obtained previously we define the relaxation of $P(U, \hat{w}, (\hat{p}^s))$ as
SP$(U, \hat{w}, (\hat{p}^s))$:

$$\begin{aligned} \text{maximize} \quad & \sum_{j\in U} \bar{p}_j(\lambda^\dagger) x_j + \sum_{s=1}^{S} \lambda_s^\dagger \hat{p}^s \\ \text{subject to} \quad & \sum_{j\in U} w_j x_j \leqslant c - \hat{w}, \\ & 0 \leqslant x_j \leqslant 1, \quad \forall j \in U, \end{aligned}$$

and its optimal solution $\bar{x}(U, \hat{w}, (\hat{p}^s))$ with the corresponding objective value $\bar{z}(U, \hat{w}, (\hat{p}^s))$. This is a continuous knapsack problem which is easily solved. If these problems are infeasible, we define $z^\star(U, \hat{w}, (\hat{p}^s)) := -\infty$ and $\bar{z}(U, \hat{w}, (\hat{p}^s)) := -\infty$, respectively. Then, a branch-and-bound algorithm can be constructed as follows.

```
Algorithm B_and_B($U, \hat{w}, (\hat{p}^s)$)
  Step 1: (Evaluate the current subproblem) Solve SP($U, \hat{w}, (\hat{p}^s)$) and obtain $\bar{x}(U, \hat{w}, (\hat{p}^s))$ and $\bar{z}(U, \hat{w}, (\hat{p}^s))$.
  Step 2: (Inprospective subproblem) If $\bar{z}(U, \hat{w}, (\hat{p}^s)) \leqslant z^\star_{\text{opt}}$, return.
  Step 3: (Feasible solution) If $\bar{x}(U, \hat{w}, (\hat{p}^s))$ is feasible to $P(U, \hat{w}, (\hat{p}^s))$, i.e., if this is a 0–1 solution, and
          $z^\star_{\text{opt}} < z(\bar{x}(U, \hat{w}, (\hat{p}^s)))$, update $z^\star_{\text{opt}} := z(\bar{x}(U, \hat{w}, (\hat{p}^s)))$.
  Step 4: (Leaf node) If $U = \emptyset$, return.
  Step 5: (Branch and recursive call) Do the following.
            (i) Find $l := \min\{j \mid j \in U\}$.
           (ii) Call B_and_B($U \backslash \{l\}, \hat{w}, (\hat{p}^s)$).
          (iii) If $\hat{w} + w_l \leqslant c$ call B_and_B($U \backslash \{l\}, \hat{w} + w_l, (\hat{p}^s + p^s_l)$).
           (iv) return.
```

Initially we call B_and_B with the parameters (13) and (14) and the *incumbent* value $z^\star_{\text{opt}} := -\infty$; and upon termination of this obtain the optimal objective value as $z^\star_{\text{opt}}$. By assumption $\mathbf{B}_1$ and the definition of l above, branching is made in the non-increasing order of $\bar{p}_j / w_j$ among the unfixed variables. Also, by the recursive nature of the algorithm, subproblems (or branch-and-bound *nodes* (BBNs)) are generated and examined in a *depth-first* fashion.

The characteristic features of this algorithm are as follows. First, we apply the pegging test and reduce the problem size before resorting to the branch-and-bound method. Second, in SP($U, \hat{w}, (\hat{p}^s)$) variable x_j is continuously relaxed. Although this may give a slightly weaker upper bound than the other relaxations where x_j is kept to be 0–1 variable, in computation time the former is much faster. And finally, in B_and_B we use the same $\lambda^\dagger$ throughout all subproblems. This is in sharp contrast to the previous methods, where λ is reoptimized from scratch at each node. By doing so, our algorithm can process each node much faster than other algorithms, and although we suffer from the increased number of subproblems generated, in total computing time we expect our method advantageous to earlier methods.

5. Numerical experiments

5.1. Design of experiments

For MKP with $n = 200, 400, \ldots, 1000$ and $S = 10, 20, 30$ we evaluate the performance of our method consisting of the surrogate relaxation, pegging and B_and_B stated in previous sections. We call this method PEG-BAB.

In preparing test problems, we follow Yu [6] as follows. First, the weight w_j and the *nominal* value p^0_j of item j are determined as independent and uniform random integers over $[1, 100]$. Then, the value under scenario s is given by

$$p^s_j : \text{uniform random integer over } [(1-\delta)p^0_j, (1+\delta)p^0_j],$$

where δ is a parameter to control the degree of correlation between different scenarios. Note that the profits are more strongly correlated for smaller δ, and we examine the cases of $\delta = 0.3, 0.6$ and 0.9. Knapsack capacity is set to

$$c := \sum_{j=1}^{n} w_j / m,$$

where m is either 2, 3 or 4.

We have implemented the algorithm for PEG-BAB in ANSI C language on an IBM RS/6000 SP44 Model 270 workstation (CPU: POWER 3-II, 375 MHz). We also solved small instances using NUOPT Version 3.3.0 (Mathematical Systems Inc., 2002) [18] on the same machine. However, the latter was only able to solve problems of relatively small size.

Table 1
Lower and upper bounds with pegging test ($\delta = 0.3$)

S	m	n	$\bar{z}$	$\underline{z}$	gap	rerr%	n'	reduc%	CPU_1
10	2	200	7801.6	7794.1	7.5	0.10	34.1	83.0	0.04
		400	15 831.0	15 823.0	8.0	0.05	67.9	83.0	0.13
		600	23 682.8	23 673.7	9.1	0.04	105.3	82.5	0.19
		800	31 744.1	31 736.3	7.8	0.02	127.4	84.1	0.44
		1000	40 323.8	40 315.1	8.7	0.02	179.7	82.0	0.50
	3	200	6381.0	6369.2	11.8	0.19	48.3	75.9	0.06
		400	13 079.3	13 070.7	8.6	0.07	66.9	83.3	0.15
		600	19 477.2	19 467.0	10.2	0.05	122.3	79.6	0.37
		800	26 071.6	26 058.8	12.8	0.05	197.2	75.3	0.75
		1000	33 188.8	33 175.0	13.8	0.04	261.0	73.9	0.82
	4	200	5487.3	5476.8	10.5	0.19	39.7	82.0	0.05
		400	11 351.6	11 340.9	10.7	0.09	69.4	82.7	0.14
		600	16 879.7	16 869.4	10.3	0.06	107.8	82.0	0.32
		800	22 583.3	22 571.5	11.8	0.05	164.4	79.5	0.79
		1000	28 775.9	28 763.7	12.2	0.04	201.7	79.8	0.90
20	2	200	7753.5	7747.2	6.3	0.08	28.9	85.6	0.10
		400	15 948.8	15 940.8	8.0	0.05	64.5	83.9	0.24
		600	23 841.0	23 833.7	7.3	0.03	95.4	84.1	0.49
		800	31 633.0	31 624.7	8.3	0.03	134.2	83.2	0.78
		1000	40 359.4	40 350.2	9.2	0.02	183.0	81.7	1.16
	3	200	6356.1	6344.9	11.2	0.18	42.6	78.7	0.11
		400	13 134.7	13 125.2	9.5	0.07	72.9	81.8	0.36
		600	19 603.2	19 590.3	12.9	0.07	151.1	74.8	0.74
		800	25 992.8	25 983.7	9.1	0.04	141.9	82.2	0.91
		1000	33 163.9	33 152.5	11.4	0.03	219.2	78.1	1.35
	4	200	5474.2	5463.9	10.3	0.19	38.2	80.9	0.11
		400	11 380.3	11 369.5	10.8	0.10	72.0	82.0	0.37
		600	16 997.5	16 985.9	11.6	0.07	119.8	80.0	0.78
		800	22 483.9	22 472.4	11.5	0.05	159.6	80.0	0.97
		1000	28 719.3	28 706.0	13.3	0.05	228.0	77.2	1.66
30	2	200	7914.0	7905.1	8.9	0.11	35.2	82.4	0.14
		400	15 780.7	15 772.3	8.4	0.05	67.4	83.2	0.36
		600	23 583.4	23 576.6	6.8	0.03	89.1	85.2	0.81
		800	31 870.2	31 862.5	7.7	0.02	126.6	84.2	1.31
		1000	39 778.8	39 770.6	8.2	0.02	163.8	83.6	1.88
	3	200	6514.8	6501.7	13.1	0.20	48.7	75.7	0.19
		400	12 966.5	12 954.5	12.0	0.09	93.6	76.6	0.66
		600	19 463.7	19 452.9	10.8	0.06	124.9	79.2	1.04
		800	26 164.3	26 151.1	13.2	0.05	202.0	74.8	1.69
		1000	32 624.8	32 613.3	11.5	0.04	225.1	77.5	2.26
	4	200	5638.2	5627.2	11.0	0.20	38.8	80.6	0.19
		400	11 235.3	11 222.6	12.7	0.11	90.3	77.4	0.52
		600	16 877.4	16 866.9	10.5	0.06	108.2	82.0	1.27
		800	22 636.0	22 624.0	12.0	0.05	166.9	79.1	1.71
		1000	28 200.6	28 187.6	13.0	0.05	227.2	77.3	2.40

5.2. Bounds and reduction

Tables 1–3 give the results of computation of surrogate relaxation and pegging. In addition to the bounds $\bar{z}$ and $\underline{z}$, the tables include

gap	the gap between the bounds $:= \bar{z} - \underline{z}$,
rerr%	relative error in percent $:= 100 \cdot (\bar{z} - \underline{z})/\underline{z}$,

Table 2
Lower and upper bounds with pegging test ($\delta = 0.6$)

S	m	n	$\bar{z}$	$\underline{z}$	gap	rerr%	n'	reduc%	CPU_1
10	2	200	7789.2	7771.5	17.7	0.23	72.3	63.9	0.11
		400	15 769.2	15 749.8	19.4	0.12	149.1	62.7	0.30
		600	23 655.5	23 633.4	22.1	0.09	253.6	57.7	0.48
		800	31 809.0	31 791.7	17.3	0.05	272.9	65.9	0.82
		1000	40 362.1	40 344.6	17.5	0.04	346.3	65.4	1.00
	3	200	6392.1	6369.8	22.3	0.35	81.4	59.3	0.11
		400	13 066.5	13 042.3	24.2	0.19	167.2	58.2	0.32
		600	19 549.5	19 527.3	22.2	0.11	241.0	59.8	0.65
		800	26 237.4	26 212.6	24.8	0.09	349.7	56.3	0.95
		1000	33 255.3	33 228.9	26.4	0.08	462.2	53.8	1.24
	4	200	5487.4	5461.9	25.5	0.47	83.6	58.2	0.11
		400	11 357.2	11 332.1	25.1	0.22	157.1	60.7	0.33
		600	16 954.4	16 926.8	27.6	0.16	263.7	56.1	0.68
		800	22 736.0	22 712.6	23.4	0.10	299.1	62.6	1.01
		1000	28 811.9	28 790.2	21.7	0.08	350.0	65.0	1.35
20	2	200	7651.0	7634.8	16.2	0.21	69.2	65.4	0.17
		400	15 833.5	15 816.6	16.9	0.11	136.4	65.9	0.63
		600	23 690.0	23 671.5	18.5	0.08	221.7	63.1	0.88
		800	31 478.5	31 460.8	17.7	0.06	289.5	63.8	1.73
		1000	40 261.8	40 239.0	22.8	0.06	423.9	57.6	2.04
	3	200	6302.6	6280.7	21.9	0.35	81.5	59.3	0.19
		400	13 088.5	13 066.4	22.1	0.17	158.0	60.5	0.72
		600	19 480.6	19 452.1	28.5	0.15	299.4	50.1	1.45
		800	25 924.4	25 897.4	27.0	0.10	378.3	52.7	2.02
		1000	33 129.4	33 101.1	28.3	0.09	491.6	50.8	2.46
	4	200	5436.5	5412.7	23.8	0.44	78.6	60.7	0.22
		400	11 347.0	11 321.5	25.5	0.23	160.5	59.8	0.68
		600	16 861.7	16 832.5	29.2	0.17	282.4	52.9	1.12
		800	22 439.6	22 411.7	27.9	0.12	359.1	55.1	2.01
		1000	28 698.4	28 666.6	31.8	0.11	497.1	50.3	2.40
30	2	200	7763.3	7747.7	15.6	0.20	65.3	67.4	0.28
		400	15 517.4	15 499.6	17.8	0.11	142.4	64.4	0.87
		600	23 368.5	23 349.2	19.3	0.08	230.6	61.6	1.45
		800	31 643.4	31 628.3	15.1	0.05	236.1	70.5	2.03
		1000	39 637.7	39 617.6	20.1	0.05	384.3	61.6	3.23
	3	200	6412.3	6392.8	19.5	0.31	72.2	63.9	0.33
		400	12 783.8	12 762.1	21.7	0.17	158.5	60.4	0.96
		600	19 311.6	19 282.0	29.6	0.15	310.3	48.3	1.87
		800	26 015.5	25 987.2	28.3	0.11	387.7	51.5	2.67
		1000	32 522.0	32 490.4	31.6	0.10	540.8	45.9	3.78
	4	200	5548.6	5528.2	20.4	0.37	71.2	64.4	0.34
		400	11 062.6	11 037.5	25.1	0.23	167.5	58.1	0.82
		600	16 744.4	16 714.4	30.0	0.18	284.4	52.6	1.79
		800	22 492.0	22 460.8	31.2	0.14	384.9	51.9	2.57
		1000	28 080.2	28 047.6	32.6	0.12	512.2	48.8	4.31

n'	the number of unfixed variables,
reduc%	the ratio of reduction in percent $=100 \cdot (n - n')/n$,
CPU_1	the time in seconds for surrogation and pegging.

Each row is the average over 10 randomly generated instances. From these tables, we observe the following:

1. By the surrogate relaxation we obtain an upper bound and a heuristic solution of high precision in a few CPU seconds. The gap between these bounds are at most 1.0% of the lower bound in the examined instances.

Table 3
Lower and upper bounds with pegging test ($\delta = 0.9$)

S	m	n	$\bar{z}$	$\underline{z}$	gap	rerr%	n'	reduc%	CPU$_1$
10	2	200	7727.1	7708.0	19.1	0.25	77.6	61.2	0.15
		400	15 627.2	15 602.6	24.6	0.16	190.0	52.5	0.31
		600	23 616.6	23 584.3	32.3	0.14	358.3	40.3	0.72
		800	31 648.0	31 619.7	28.3	0.09	425.7	46.8	1.11
		1000	40 511.1	40 482.2	28.9	0.07	521.2	47.9	1.47
	3	200	6355.1	6320.6	34.5	0.55	114.4	42.8	0.15
		400	13 008.1	12 978.8	29.3	0.23	198.0	50.5	0.35
		600	19 540.1	19 500.2	39.9	0.20	383.1	36.2	0.78
		800	26 178.3	26 139.8	38.5	0.15	500.6	37.4	0.98
		1000	33 469.0	33 429.2	39.8	0.12	634.8	36.5	1.69
	4	200	5481.5	5444.0	37.5	0.69	115.6	42.2	0.15
		400	11 309.3	11 274.5	34.8	0.31	208.1	48.0	0.44
		600	16 937.8	16 899.0	38.8	0.23	343.5	42.8	0.79
		800	22 674.3	22 633.9	40.4	0.18	471.9	41.0	1.08
		1000	28 984.5	28 946.7	37.8	0.13	557.2	44.3	1.57
20	2	200	7573.3	7545.4	27.9	0.37	110.0	45.0	0.27
		400	15 675.0	15 648.7	26.3	0.17	198.1	50.5	0.69
		600	23 494.0	23 461.1	32.9	0.14	358.6	40.2	1.07
		800	31 485.6	31 452.6	33.0	0.10	467.5	41.6	2.28
		1000	40 107.5	40 073.1	34.4	0.09	606.5	39.4	2.66
	3	200	6249.1	6217.0	32.1	0.52	111.0	44.5	0.33
		400	12 972.8	12 936.6	36.2	0.28	241.1	39.7	0.78
		600	19 424.8	19 387.2	37.6	0.19	373.5	37.8	1.34
		800	26 029.5	25 977.2	52.3	0.20	613.9	23.2	2.76
		1000	33 140.5	33 090.5	50.0	0.15	728.8	27.1	3.62
	4	200	5395.8	5361.4	34.4	0.64	106.8	46.6	0.30
		400	11 236.2	11 197.9	38.3	0.34	223.2	44.2	0.83
		600	16 836.2	16 797.4	38.8	0.23	346.4	42.3	1.60
		800	22 520.3	22 474.7	45.6	0.20	520.4	35.0	2.73
		1000	28 721.5	28 675.4	46.1	0.15	654.6	34.5	3.28
30	2	200	7708.1	7677.2	30.9	0.40	113.9	43.1	0.41
		400	15 260.1	15 233.5	26.6	0.17	209.8	47.6	1.10
		600	23 145.4	23 112.8	32.6	0.14	357.7	40.4	2.33
		800	31 487.7	31 455.0	32.7	0.10	464.5	41.9	3.21
		1000	39 308.2	39 268.7	39.5	0.10	680.1	32.0	3.91
	3	200	6357.2	6324.5	32.7	0.52	112.1	44.0	0.44
		400	12 606.5	12 565.1	41.4	0.33	269.9	32.5	1.29
		600	19 141.2	19 096.1	45.1	0.24	422.8	29.5	2.65
		800	25 935.4	25 881.8	53.6	0.21	621.3	22.3	3.83
		1000	32 318.8	32 262.4	56.4	0.17	801.7	19.8	5.38
	4	200	5474.2	5432.7	41.5	0.76	120.6	39.7	0.49
		400	10 917.1	10 871.6	45.5	0.42	266.7	33.3	1.50
		600	16 578.5	16 536.1	42.4	0.26	376.2	37.3	2.44
		800	22 426.6	22 373.2	53.4	0.24	589.5	26.3	3.83
		1000	27 915.0	27 855.5	59.5	0.21	782.9	21.7	5.27

2. The effectiveness of the pegging test (as measured by 'reduc%') is larger for the smaller value of δ. This is because the gap and n' increase with δ.
3. CPU$_1$ increases with n and S, but this is rather insensitive to δ and m. In all cases tested, this takes at most a few seconds.
4. The objective value ($\bar{z}$ and $\underline{z}$) increases with n and decreases with m, but it remains almost constant with respect to δ.

Table 4
Exact solution by PEG-BAB ($\delta = 0.3$)

S	m	n	$z^\star$	BBN ($\times 10^6$)	CPU$_2$	CPU$_\mathrm{T}$	#sol
10	2	200	7797.7	0.0	0.00	0.05	10
		400	15 827.6	0.2	0.15	0.28	10
		600	23 680.8	0.1	0.08	0.27	10
		800	31 741.7	2.3	1.64	2.08	10
		1000	40 322.2	0.6	0.45	0.95	10
	3	200	6375.6	0.0	0.03	0.09	10
		400	13 075.2	1.0	0.74	0.89	10
		600	19 473.5	20.1	14.67	15.04	10
		800	26 067.4	105.1	75.99	76.74	10
		1000	33 185.1	240.1	175.68	176.50	10
	4	200	5484.0	0.0	0.01	0.07	10
		400	11 346.0	0.5	0.37	0.52	10
		600	16 875.1	41.3	29.88	30.20	10
		800	22 578.4	211.8	150.75	151.54	9
		1000	28 772.5	57.9	40.85	41.75	10
20	2	200	7749.3	0.0	0.01	0.11	10
		400	15 944.5	0.1	0.15	0.40	10
		600	23 837.4	0.6	0.62	1.12	10
		800	31 630.9	1.5	1.53	2.31	10
		1000	40 357.0	121.9	124.07	125.22	10
	3	200	6350.4	0.1	0.07	0.17	10
		400	13 129.6	1.9	2.02	2.38	10
		600	19 598.1	37.9	39.10	39.84	10
		800	25 988.5	165.6	171.00	171.91	9
		1000	33 159.8	449.9	461.83	463.18	8
	4	200	5467.9	0.1	0.08	0.19	10
		400	11 376.5	1.4	1.50	1.87	10
		600	16 991.6	208.6	216.23	217.01	9
		800	22 478.6	312.6	322.10	323.06	8
		1000	28 714.4	505.8	519.74	521.40	8
30	2	200	7909.1	0.0	0.02	0.16	10
		400	15 777.7	0.1	0.17	0.53	10
		600	23 580.7	0.5	0.63	1.44	10
		800	31 867.1	10.9	14.60	15.91	10
		1000	39 775.4	177.0	239.78	241.65	9
	3	200	6506.7	0.4	0.55	0.74	10
		400	12 959.8	54.9	74.27	74.93	10
		600	19 459.7	38.7	52.23	53.28	10
		800	26 159.5	300.9	408.52	410.20	8
		1000	32 619.1	463.9	630.33	632.59	5
	4	200	5630.6	0.1	0.08	0.27	10
		400	11 229.1	1.9	2.55	3.07	10
		600	16 872.4	72.0	98.74	100.01	10
		800	22 630.3	328.7	442.47	444.18	7
		1000	28 194.8	355.9	486.06	488.46	7

5.3. Exact solution

Tables 4–6 show the results of computation of exact solutions. For each value of δ, S, m and n, we computed the same 10 instances as in Section 5.2, and the average of these are shown in the tables. We truncated B_and_B at the time limit of TL = 1200 s. Here, $z^\star$ is the optimal objective value; and in case that the computation is truncated at TL, this shows the best objective value obtained up to that time. The '#sol' and 'BBN' indicate the number of instances (out of 10) solved completely within TL and the number of the generated BBNs in millions, respectively. Computation

Table 5
Exact solution by PEG-BAB ($\delta = 0.6$)

S	m	n	$z^\star$	BBN ($\times 10^6$)	CPU_2	CPU_T	#sol
10	2	200	7781.7	2.1	1.54	1.66	10
		400	15 762.9	7.9	5.64	5.94	10
		600	23 649.8	55.8	39.50	39.98	10
		800	31 803.7	403.4	291.54	292.36	8
		1000	40 357.6	890.2	639.32	640.32	5
	3	200	6380.7	5.6	4.05	4.16	10
		400	13 057.6	120.6	86.84	87.16	10
		600	19 539.2	1079.9	770.71	771.35	6
		800	26 227.0	1338.1	968.93	969.88	3
		1000	33 246.2	1477.3	1082.71	1083.95	1
	4	200	5473.7	8.3	6.07	6.19	10
		400	11 346.1	239.3	171.87	172.21	9
		600	16 943.0	1118.0	807.43	808.11	6
		800	22 724.7	1207.1	877.53	878.54	4
		1000	28 802.3	1250.5	912.87	914.21	3
20	2	200	7641.7	35.2	38.45	38.62	10
		400	15 826.0	125.3	131.01	131.64	10
		600	23 682.6	357.9	366.18	367.06	8
		800	31 469.3	737.3	757.77	759.50	5
		1000	40 252.2	890.4	929.58	931.62	3
	3	200	6291.9	5.6	5.81	6.00	10
		400	13 076.1	474.4	500.39	501.10	7
		600	19 467.6	1000.7	1039.07	1040.52	2
		800	25 908.5	1146.0	1198.01	1200.03	0
		1000	33 114.4	1132.6	1197.57	1200.03	0
	4	200	5422.6	20.4	21.38	21.60	10
		400	11 334.1	558.1	585.07	585.75	7
		600	16 847.3	1169.2	1198.91	1200.02	0
		800	22 420.6	1151.2	1198.00	1200.01	0
		1000	28 678.1	1125.9	1197.62	1200.02	0
30	2	200	7753.1	0.8	1.10	1.38	10
		400	15 510.8	82.6	109.62	110.49	10
		600	23 360.2	482.5	650.87	652.32	5
		800	31 635.2	563.5	767.31	769.34	5
		1000	39 625.8	841.5	1148.42	1151.65	1
	3	200	6399.1	9.1	12.53	12.86	10
		400	12 772.0	300.0	399.49	400.45	8
		600	19 295.1	793.1	1085.84	1087.72	2
		800	25 995.6	868.4	1197.35	1200.02	0
		1000	32 500.4	864.4	1196.23	1200.01	0
	4	200	5533.1	25.1	34.97	35.31	10
		400	11 049.1	402.3	534.35	535.17	7
		600	16 726.2	848.5	1154.60	1156.39	1
		800	22 473.1	868.4	1197.46	1200.02	0
		1000	28 056.5	863.6	1195.74	1200.05	0

time is shown as CPU_2, indicating the time needed to solve the reduced problem by B_and_B, and the total time $\mathrm{CPU}_\mathrm{T} = \mathrm{CPU}_1 + \mathrm{CPU}_2$ to solve the problem completely. All the rows, except for the column of #sol, are the average of these values over 10 instances.

We observe the following from these tables:

1. For $\delta = 0.3$, most of the instances with $n \leqslant 1000$, $S \leqslant 30$ and $m \leqslant 4$ were solved exactly within a few minutes.
2. For $\delta \geqslant 0.6$, the problems are sometimes hard to solve within TL(=1200 s), especially with the increase of δ, S and n. Even in such a case, most problems with $n \leqslant 200$ were solved successfully within a few minutes.

Table 6
Exact solution by PEG-BAB ($\delta = 0.9$)

S	m	n	$z^\star$	BBN $(\times 10^6)$	CPU_2	CPU_T	#sol
10	2	200	7716.6	9.3	6.84	7.00	10
		400	15 618.3	207.8	148.72	149.03	9
		600	23 606.4	1572.3	1123.01	1123.73	2
		800	31 635.2	1407.4	1019.89	1020.99	2
		1000	40 499.5	1539.2	1133.88	1135.35	1
	3	200	6339.3	209.5	150.62	150.76	9
		400	12 994.7	597.1	429.71	430.06	9
		600	19 524.1	1646.9	1199.24	1200.02	0
		800	26 159.5	1630.2	1199.04	1200.02	0
		1000	33 448.5	1598.7	1198.32	1200.01	0
	4	200	5464.0	56.5	41.36	41.51	10
		400	11 292.9	770.3	553.14	553.58	7
		600	16 918.3	1498.6	1091.83	1092.62	1
		800	22 657.5	1633.0	1198.94	1200.02	0
		1000	28 963.4	1607.9	1198.45	1200.02	0
20	2	200	7559.9	201.4	207.49	207.76	9
		400	15 663.0	679.1	702.56	703.24	6
		600	23 480.0	1047.7	1090.42	1091.49	1
		800	31 461.8	1144.8	1197.72	1200.01	0
		1000	40 089.2	1132.4	1197.35	1200.01	0
	3	200	6230.5	371.0	382.98	383.30	7
		400	12 953.3	999.6	1036.86	1037.64	3
		600	19 399.9	1145.3	1198.68	1200.02	0
		800	25 988.8	1121.6	1197.25	1200.02	0
		1000	33 106.6	1110.7	1196.39	1200.01	0
	4	200	5374.7	289.4	302.63	302.93	8
		400	11 215.8	924.8	969.03	969.86	2
		600	16 808.7	1145.3	1198.41	1200.01	0
		800	22 485.8	1118.5	1197.29	1200.01	0
		1000	28 683.3	1116.7	1196.74	1200.02	0
30	2	200	7691.6	61.9	84.20	84.61	10
		400	15 247.5	733.7	975.29	976.39	2
		600	23 126.7	882.3	1197.71	1200.04	0
		800	31 464.5	870.3	1196.81	1200.02	0
		1000	39 279.4	871.6	1196.11	1200.01	0
	3	200	6337.5	76.5	102.64	103.07	10
		400	12 581.7	811.5	1084.78	1086.07	1
		600	19 105.8	870.4	1197.36	1200.01	0
		800	25 897.2	853.2	1196.18	1200.01	0
		1000	32 270.9	860.4	1194.64	1200.01	0
	4	200	5448.7	414.6	555.82	556.30	7
		400	10 889.1	890.1	1198.52	1200.03	0
		600	16 547.0	872.0	1197.58	1200.01	0
		800	22 382.8	861.3	1196.18	1200.02	0
		1000	27 868.5	852.5	1194.74	1200.01	0

3. Even if the computation is truncated at TL, we usually obtain an approximate solution which is very close to the optimal $z^\star$.

5.4. Comparison against previous methods

Table 7 compares the previous computation of MKP [6,10] against PEG-BAB. Here we fix $n = 60$, and for each values of δ, S and m, average number of BBNs and CPU time in seconds over 10 randomly generated instances are shown. Due to the difference of the random number generators used, the instances reported in Yu and Iida are not

Table 7
Comparison against previous works ($n = 60$)

δ	S	m	Yu[a]		Iida[b]		PEG-BAB[c]	
			BBN	CPU	BBN	CPU	BBN	CPU
0.3	10	2	43.0	227.5 (81.3)	190.78	3.0 (21.4)	416.4	0.007
		3	36.4	200.8 (50.2)	276.12	3.6 (18.0)	977.2	0.010
		4	29.6	120.1 (33.4)	231.44	3.0 (16.7)	904.5	0.009
	20	2	42.4	217.9 (38.9)	256.62	6.0 (21.4)	542.1	0.014
		3	35.4	234.3 (30.8)	321.24	7.3 (19.2)	1088.2	0.019
		4	29.8	200.7 (22.8)	295.82	6.3 (14.3)	1276.7	0.022
	30	2	40.8	221.5 (22.2)	255.00	8.1 (16.2)	773.0	0.025
		3	36.0	229.2 (16.4)	320.94	9.6 (13.7)	1469.9	0.035
		4	29.8	190.2 (14.4)	274.04	7.9 (12.0)	1543.4	0.033
0.6	10	2	43.7	231.3 (34.0)	341.70	4.9 (14.4)	2650.6	0.017
		3	36.2	270.8 (27.1)	494.18	6.8 (13.6)	7140.3	0.025
		4	30.0	204.9 (19.0)	442.96	5.7 (10.6)	9142.2	0.027
	20	2	42.2	245.5 (17.0)	406.60	9.3 (12.9)	7407.3	0.036
		3	35.0	243.3 (9.5)	626.38	13.2 (10.3)	23 024.1	0.064
		4	33.0	212.1 (8.7)	428.56	8.6 (7.0)	21 159.1	0.061
	30	2	43.2	241.3 (9.7)	465.30	13.8 (11.1)	10 857.5	0.062
		3	35.8	227.6 (5.9)	696.72	20.6 (10.6)	24 462.2	0.097
		4	31.6	191.2 (5.1)	577.28	15.8 (8.4)	24 377.5	0.094
0.9	10	2	44.8	334.0 (20.9)	726.48	9.1 (11.4)	20 059.3	0.040
		3	38.0	322.4 (8.8)	882.92	10.7 (5.8)	81 743.6	0.092
		4	31.0	253.2 (9.6)	1099.22	12.4 (9.4)	50 352.7	0.066
	20	2	44.2	262.4 (5.7)	965.00	20.0 (8.7)	57 073.4	0.115
		3	36.4	246.9 (2.4)	1338.56	26.0 (5.1)	179 430.0	0.254
		4	31.4	207.4 (2.3)	1187.24	21.4 (4.7)	160 361.1	0.230
	30	2	44.8	252.3 (1.8)	1290.10	36.9 (5.3)	183 361.0	0.342
		3	38.0	234.9 (0.7)	1829.58	48.0 (2.9)	526 705.5	0.834
		4	30.0	189.1 (0.6)	1557.82	38.9 (2.3)	539 307.8	0.849

The numbers inside parentheses indicate the normalized CPU time with the machine speed estimated as a:b:c = 1:20:400.
[a]On IBM 3090/220E (two CPUs, 32 Mips). Cited from [6].
[b]On SUN SPARCstation 20 Model 71 (SPECint_base95 = 2.46, SPECfp_base95 = 2.14). Cited from [10].
[c]On IBM RS/6000 SP44 Model 270 (SPECint_base95 = 24.5, SPECfp_base95 = 48.2). See [19].

exactly the same to ours. Yu's result was obtained on an IBM 3090/220E, and Iida's was on a SUN SPARCstation 20 Model 71. Iida estimates the latter 20 times faster than the former. Our IBM RS/6000 44P-279 appears to be 10–20 times faster than Iida's. Performance data of these computers [19] are given as footnotes in Table 7, and from these we estimate the speed of computers as Yu:Iida:PEG-BAB = 1:20:400. The CPU times normalized by this factor are given in parentheses in the table.

From this table we observe the following:

1. In almost all cases, PEG-BAB solves MKP with $n = 60$ faster than previous algorithms.
2. In BBN, PEG-BAB generates much more numbers of BBNs than other algorithms. This shows that the upper bound by surrogation is 'weak', compared to those developed in previous works. Also, the use of the same $\lambda^\dagger$ throughout all subproblems, instead of computing the optimal λ at each node, makes the upper bound weaker. However, this is compensated for by the quick processing of each node, where the upper bound is easily obtained by solving a continuous knapsack problem.

5.5. An alternative algorithm

From Tables 4–6, we see that the number of the BBNs generated tends to be very large, even when the required CPU time is relatively small. To make BBN smaller, we may solve $P(U, \hat{w}, (\hat{p}^s))$ exactly instead of the continuous relaxation

Table 8
Comparison of two PEG-BAB algorithms

δ	n	PEG-BAB		PEG-BAB†	
		BBN ($\times 10^3$)	CPU	BBN ($\times 10^3$)	CPU
0.3	200	3.7	0.05	0.7	0.06
	400	202.9	0.28	33.6	0.58
	600	109.8	0.27	18.4	0.47
0.6	200	2079.3	1.66	494.3	5.54
	400	7879.7	5.94	2007.5	25.99
	600	55 817.2	39.98	11 945.4	192.73

SP$(U, \hat{w}, (\hat{p}^s))$ in Step 1 of B_and_B and obtain a stronger upper bound. To solve the 0–1 knapsack problem, we plugged 'exknap', a fast C code developed by Pisinger [20], into our program, and call the resulting algorithm PEG-BAB†.

Table 8 gives a comparison between PEG-BAB and PEG-BAB† for the limited case of $S = 10$, $m = 2$ and $n = 200, 400, 600$. We solved the same 10 random instances as before, and the table shows the average of these measurements. As expected, BBN is substantially reduced in PEG-BAB†. However, in CPU time PEG-BAB usually overperforms PEG-BAB†.

6. Conclusion

We gave heuristic and exact algorithms to solve MKP. In particular, we introduced a surrogate relaxation to derive an upper bound and a lower bound very quickly. The relaxed problem is a continuous knapsack problem, and thus the pegging test originally developed for the standard 0–1 knapsack problem can also be applied to reduce the size of MKP. This relaxation can also be exploited in the branch-and-bound algorithm to obtain an upper bound quickly.

As a result, we were able to solve MKPs with $n \leqslant 1000$ and $S \leqslant 30$ approximately in less than a few seconds, and obtained solution was usually within 1.0% of relative errors. After reducing the problem size, we solved the remaining problem by B_and_B, and this method solved larger problems to optimality faster than the previously published algorithms.

Acknowledgment

The authors are grateful to an anonymous referee for careful reading of the manuscript and constructive comments.

References

[1] Martello S, Toth P. Knapsack problems: algorithms and computer implementations. New York: Wiley; 1990.
[2] Kellerer H, Pferschy U, Pisinger D. Knapsack problems. Berlin: Springer; 2004.
[3] Stauer RE. Multiple criteria optimization: theory, computation and application. New York: Wiley; 1986.
[4] Gass SI, Harris CM, editors. Encyclopedia of operations research and management science. Dordrecht: Kluwer Academic Publishers; 1996.
[5] Du DZ, Pardalos PM. Minimax and applications. Dordrecht: Kluwer Academic Publishers; 1995.
[6] Yu G. On the max–min 0–1 knapsack problem with robust optimization applications. Operations Research 1973;44:407–15.
[7] Garey MR, Johnson DS. Computers and intractability: a guide to the theory of NP-completeness. San Francisco, CA: Freeman; 1979.
[8] Fourer R. Software survey: linear programming. OR/MS Today 1999;26:64–71.
[9] Kouvelis P, Yu G. Robust discrete optimization and its applications. Dordrecht: Kluwer Academic Publishers; 1997.
[10] Iida H. A note on the max–min 0–1 knapsack problem. Journal of Combinatorial Optimization 1999;3:89–94.
[11] Glover F. Surrogate constraint duality in mathematical programming. Operations Research 1975;23:434–51.
[12] Wolsey LA. Integer programming. New York: Wiley; 1998.
[13] Fisher M. The Lagrangian relaxation method for solving integer programming problems. Management Science 2004;50:1861–71.
[14] Nemhauser GL, Wolsey LA. Integer and combinatorial optimization. New York: Wiley; 1988.
[15] Ingargiola GP, Korsh FF. Reduction algorithms for zero–one single knapsack problems. Management Science 1973;20:460–3.
[16] Fayard D, Plateau G. Resolution of the 0–1 knapsack problem: comparison of methods. Mathematical Programming 1975;8:272–307.
[17] Dembo DS, Hammer PL. A reduction algorithm for knapsack problems. Methods of Operations Research 1980;36:49–60.

[18] Mathematical Systems Inc. Nuopt version 3.3.0; 2002. ⟨http://www.msi.co.jp/nuopt⟩.
[19] SPEC; 2005. ⟨http://www.spec.org/osg/cpu95/results⟩.
[20] Pisinger D. An expanding-core algorithm for the exact 0–1 knapsack problem. European Journal of Operational Research 1995;87:175–87 Source code 'exknap' available at: ⟨http://www.diku.dk/∼pisinger⟩.