

# A virtual pegging approach to the max–min optimization of the bi-criteria knapsack problem

Fumiaki Taniguchi, Takeo Yamada\* and Seiji Kataoka

*Department of Computer Science, National Defense Academy, Yokosuka, Kanagawa, Japan*

*(Received 28 January 2007; revised version received 29 April 2007; second revision received 12 August 2007; accepted 17 September 2007)*

We are concerned with a variation of the knapsack problem, the bi-objective max–min knapsack problem (BKP), where the values of items differ under two possible scenarios. We have given a heuristic algorithm and an exact algorithm to solve this problem. In particular, we introduce a surrogate relaxation to derive upper and lower bounds very quickly, and apply the pegging test to reduce the size of BKP. We also exploit this relaxation to obtain an upper bound in the branch-and-bound algorithm to solve the reduced problem. To further reduce the problem size, we propose a ‘virtual pegging’ algorithm and solve BKP to optimality. As a result, for problems with up to 16,000 items, we obtain a very accurate approximate solution in less than a few seconds. Except for some instances, exact solutions can also be obtained in less than a few minutes on ordinary computers. However, the proposed algorithm is less effective for strongly correlated instances.

**Keywords:** knapsack problem; bi-objective combinatorial optimization; pegging test

*2000 AMS Subject Classification:* 90C27; 90C29; 49K35

## 1. Introduction

Knapsack problem [20,22] has been studied extensively in operations research and computer science. Although it is an  $\mathcal{NP}$ -hard [15] combinatorial optimization problem, it can be solved relatively easy in practice. In this article, we are concerned with a variation of this problem, where the values of items differ under possible two *scenarios*. By  $p_j^k$  we denote the value of item  $j$  under scenario  $k (= 1, 2)$ , and  $x_j$  is the decision variable that takes value 1 if item  $j$  is adopted and 0 otherwise for  $j = 1, 2, \dots, n$ . Then,

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

is the total value of the solution  $x = (x_j)$  under scenario  $k$ , and thus we have two objective functions to maximize. On the other hand, the weight of item  $j$  is assumed to be constant  $w_j$  through all scenarios, and the knapsack capacity is  $c$ .

---

\*Corresponding author. Email: yamada@nda.ac.jp

Then, we formulate the bi-objective max–min knapsack problem [4,13,16] as

BKP:

$$\text{maximize} \quad \min \left\{ \sum_{j=1}^n p_j^1 x_j, \sum_{j=1}^n p_j^2 x_j \right\} \quad (2)$$

$$\text{subject to} \quad \sum_{j=1}^n w_j x_j \leq c, \quad (3)$$

$$x_j \in \{0, 1\}, \quad j = 1, 2, \dots, n. \quad (4)$$

Without much loss of generality, we assume in the sequel that

A<sub>1</sub>:  $p_j^k$  ( $j = 1, 2, \dots, n$ ;  $k = 1, 2$ ) are non-negative integers.

A<sub>2</sub>:  $w_j$  ( $j = 1, 2, \dots, n$ ) and  $c$  are positive integers.

A<sub>3</sub>:  $\sum_{j=1}^n w_j > c$ .

By rewriting BKP as the following equivalent linear integer programming problem, we may solve small instances using free or commercial IP solvers [7].

BKP<sup>#</sup>:

$$\text{maximize} \quad v \quad (5)$$

$$\text{subject to} \quad \sum_{j=1}^n p_j^k x_j \geq v, \quad k = 1, 2, \quad (6)$$

$$(3), (4) \text{ and } v \geq 0.$$

Bi-objective knapsack problem has been studied by Eben-Chaime [4] and Zhang and Ong [13], who presented parametric or heuristic algorithms to solve the problem. Yu [3], Iida [12] and Kouvelis and Yu [16] gave branch-and-bound algorithms for the multi-scenario max–min knapsack problem (MKP), respectively, and solved problems with  $n \leq 90$  items and 30 scenarios.

In a companion paper [9], we gave reduction and exact algorithms to solve MKP with more than two scenarios. Here, we focus on the case of two scenarios, and present an algorithm that can solve much larger problems than the previous algorithms by employing a virtual pegging approach [3]. First, we introduce in Section 2 the surrogate relaxation to find upper and lower bounds quickly. Then in Section 3, following [9], we introduce a pegging test to reduce the size of the problem, and extend this to the virtual pegging test in Section 4. Through these, the original BKP is reduced (often remarkably) in size, and finally we solve the reduced problem by the ‘surrogate relaxation-based’ branch-and-bound algorithm of Section 5. Through this approach, we are often able to solve BKPs with up to 16,000 items in less than a few minutes. However, in strongly correlated instances, we frequently encounter difficulty in solving small problems with a few hundred items.

## 2. Upper and lower bounds

This section derives an upper bound by applying the surrogate relaxation [1,3] to BKP<sup>#</sup>. At the same time, we obtain an approximate solution, and thus a lower bound to BKP.

## 2.1 Surrogate relaxation

For an arbitrary  $\lambda \in [0, 1]$  we define the surrogate relaxation of BKP as follows.

SBKP( $\lambda$ ):

$$\text{maximize} \quad \sum_{j=1}^n \bar{p}_j(\lambda)x_j \quad (7)$$

$$\text{subject to} \quad \sum_{j=1}^n w_j x_j \leq c, \quad (8)$$

$$0 \leq x_j \leq 1, \quad j = 1, \dots, n, \quad (9)$$

where

$$\bar{p}_j(\lambda) := \lambda p_j^1 + (1 - \lambda)p_j^2. \quad (10)$$

Here, we note that  $x_j$  is also relaxed to a continuous variable.

For a fixed  $\lambda \in [0, 1]$ , SBKP( $\lambda$ ) is the continuous knapsack problem whose solution is easily found [20]. Let  $\bar{x}(\lambda) = (\bar{x}_j(\lambda))$  denote an optimal solution to SBKP( $\lambda$ ) with the corresponding optimal value  $\bar{z}(\lambda)$ , and  $z^*$  is the optimal objective value to the original BKP. Then, we have

$$z^* \leq \bar{z}(\lambda)$$

i.e.,  $\bar{z}(\lambda)$  gives an upper bound to BKP.

Analogous to the Lagrangian relaxation [5,19],  $\bar{z}(\lambda)$  satisfies the following properties [9].

### PROPOSITION 1

- (1)  $\bar{z}(\lambda)$  is a piecewise-linear, convex function of  $\lambda$ .
- (2) If  $\bar{z}(\lambda)$  is differentiable at  $\lambda$ ,

$$\frac{d\bar{z}(\lambda)}{d\lambda} = z^1(\bar{x}(\lambda)) - z^2(\bar{x}(\lambda)). \quad (11)$$

- (3) For  $\lambda \in [0, 1]$ , if  $\bar{x}(\lambda)$  is feasible to BKP and

$$z^1(\bar{x}(\lambda)) = z^2(\bar{x}(\lambda)), \quad (12)$$

then  $\bar{x}(\lambda)$  is an optimal solution to BKP.

## 2.2 Binary search method

For an arbitrary  $\lambda \in [0, 1]$ ,  $\bar{z}(\lambda)$  gives an upper bound to BKP. However, to find an upper bound with  $\bar{z}(\lambda)$  as small as possible, we solve the following surrogate dual problem [1]

$$\begin{aligned} &\text{minimize} \quad \bar{z}(\lambda) \\ &\text{subject to} \quad \lambda \in [0, 1]. \end{aligned}$$

Then, taking Equation (11) into account, the following binary search method solves the dual problem.

ALGORITHM BINARY

- Step 1 Let  $\lambda_L := 0$  and  $\lambda_R := 1$ .
- Step 2 Let  $\lambda := (\lambda_L + \lambda_R)/2$  and solve SBKP( $\lambda$ ) to obtain  $\bar{x}(\lambda)$  and  $\bar{z}(\lambda)$ .
- Step 3 If  $\lambda_R - \lambda_L < \epsilon$ , or the condition in Equation (12) is met, go to Step 5.
- Step 4 If  $z^1(\bar{x}(\lambda)) > z^2(\bar{x}(\lambda))$  let  $\lambda_R := \lambda$ , else let  $\lambda_L := \lambda$ . Go to Step 2.
- Step 5 Output  $\bar{x}(\lambda)$  and  $\bar{z}(\lambda)$ , and stop.

Here  $\epsilon$  is a sufficiently small ‘tolerance limit’ of computation, and by  $\lambda^\dagger$  we denote  $\lambda$  upon termination of the above algorithm. Thus, we obtain an optimal upper bound to BKP as  $\bar{z} := \bar{z}(\lambda^\dagger)$ .

2.3 Lower bounds

For an arbitrary  $\lambda \in [0, 1]$ ,  $\bar{x}(\lambda)$  satisfies Equation (3). If this also satisfies the 0–1 constraint (4), this is feasible to BKP; hence, the corresponding objective value gives a lower bound to the original problem. If, on the other hand, some components of  $\bar{x}(\lambda)$  violate Equation (4), we still obtain a feasible solution by replacing all the fractional components with 0. In BINARY, each time we solve SBKP( $\lambda$ ) we thus get a lower bound, and the largest one found this way gives the best lower bound. This is henceforth denoted as  $\underline{z}$ .

3. Pegging test

Pegging test [2,10,18] is well known for the ordinary 0–1 knapsack problem. By applying this test, some variables are fixed either at 0 or 1, and after removing these we obtain a problem of (often significantly) reduced size. In this section, we show that the same pegging test can be applied to BKP by introducing the surrogate relaxation first, as we have shown in [9] for MKP in general.

Assume that we have the optimal surrogate multiplier  $\lambda^\dagger$ , the corresponding upper bound  $\bar{z} = \bar{z}(\lambda^\dagger)$  and a lower bound  $\underline{z}$  to BKP, and let us consider SBKP( $\lambda^\dagger$ ). In what follows, we write  $\bar{p}_j := \bar{p}_j(\lambda^\dagger)$  for simplicity. For an arbitrary  $u \in \{1, 2, \dots, n\}$ , let  $\bar{z}_{u,\delta}$  denote the optimal objective value to SBKP( $\lambda^\dagger$ ) with an additional constraint  $x_u = \delta$ , where  $\delta$  is either 0 or 1. Then, if

$$\bar{z}_{u,0} < \underline{z} \tag{13}$$

it is not possible that  $x_u^* = 0$  in any optimal solution  $x^* = (x_j^*)$  to BKP, i.e., we necessarily have  $x_u^* = 1$ . Similarly, in the case that

$$\bar{z}_{u,1} < \underline{z} \tag{14}$$

$x_u^* = 0$  must follow.

To determine Equation (13) and (14) quickly, the following shortcut is usually taken. First of all, without loss of generality, we assume the following.

**B<sub>1</sub>**: The items are numbered in the non-increasing order of  $\bar{p}_j/w_j$ .

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

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

where  $W_0 = P_0 = 0$ . Then, the broken line connecting  $\{(W_j, P_j) | j = 0, \dots, n\}$  gives a piecewise-linear, monotonically non-decreasing, concave function [20].

The intersection of this broken line with the vertical line  $W = c$  gives an upper bound  $\bar{z}$ . The item  $s$  satisfying  $W_{s-1} \leq c < W_s$  is said to be the critical item. Here, if for any  $u < s$  we set  $x_u = 0$ , it is known [2,18] that

$$\bar{z}_{u,0} \leq \bar{z} - \theta_u, \tag{15}$$

where we define

$$\theta_u := \bar{p}_u - \left( \frac{\bar{p}_s}{w_s} \right) w_u. \tag{16}$$

This is referred to as the threshold for item  $u$ . Then, we have the following [9].

**THEOREM 2** For any optimal solution  $x^* = (x_j^*)$  to BKP, both of the following hold.

- (1)  $\bar{z} - \underline{z} < \theta_j \Rightarrow x_j^* = 1$ , and
- (2)  $\bar{z} - \underline{z} < -\theta_j \Rightarrow x_j^* = 0$ .

For a pair of upper and lower bounds, by applying this theorem some variables are fixed, and removing these variables we obtain a BKP of (often significantly) reduced size.

### 4. Virtual pegging test

In Theorem 1, we see that the smaller the gap ( $= \bar{z} - \underline{z}$ ) between the upper and lower bounds, the more variables are fixed. If the gap is not small enough, the effectiveness of the pegging method is limited, since the size of the problem will not be reduced much in such a case. In the present section, we introduce a virtual pegging test [17], **which we originally presented for the precedence constrained knapsack problem, to BKP** to cope with this problem.

#### 4.1 Virtual pegging principle

In the pegging test based on Theorem 1, the upper and lower bounds necessarily satisfy

$$\underline{z} \leq z^* \leq \bar{z}.$$

However, we may carry out this test using an arbitrary value  $l \leq \bar{z}$  as an ‘assumed’ lower bound. Let the set of all the feasible solutions to BKP be  $X$ , i.e.,

$$X := \left\{ (x_1, x_2, \dots, x_n) \mid \sum_{j=1}^n w_j x_j \leq c, x_j \in \{0, 1\}, \forall j \right\}.$$

Then, if we carry out the pegging test (Theorem 1) with  $\bar{z}$  and  $l$  as upper and lower bounds, some  $x_j$ ’s will be ‘fixed’ either at 0 or 1. But this is not necessarily a correct pegging because  $l$  is not a guaranteed lower bound to BKP. Let the index sets of variables, which are (temporally) fixed at 0 and 1 by Theorem 1, be  $F_0(l)$ , and  $F_1(l)$  respectively. Then, we have the following reduced problem.

Downloaded By: [Yamada, Takeo] At: 02:45 12 June 2009

$R(l)$

$$\text{maximize} \quad \min \left\{ \sum_{j=1}^n p_j^1 x_j, \sum_{j=1}^n p_j^2 x_j \right\} \quad (17)$$

subject to  $x \in X,$

$$x_j = 0, \quad \forall j \in F_0(l), \quad (18)$$

$$x_j = 1, \quad \forall j \in F_1(l). \quad (19)$$

The optimal objective value to this problem will be denoted as  $z_l^*$ . If  $R(l)$  is infeasible, we define  $z_l^* := -\infty$ . Then, for the optimal objective value  $z^*$  to BKP we have the following [17].

THEOREM 3

- (1)  $l \leq z^* \Rightarrow z_l^* = z^*,$
- (2)  $l > z^* \Rightarrow z_l^* \leq z^*,$
- (3)  $l \leq l' \Rightarrow z_l^* \geq z_{l'}^*,$
- (4)  $l \leq z_l^* \Rightarrow z_l^* = z^*.$

As a direct corollary to (3), if  $R(l)$  is infeasible, then  $R(l')$  is also infeasible for all  $l' \geq l$ .

## 4.2 A virtual pegging algorithm

For an arbitrary value  $l \leq \bar{z}$ , by carrying out the virtual pegging test and solving the reduced problem  $R(l)$ , we obtain  $z_l^*$ . Then, if (4) is satisfied in Theorem 2, BKP is solved. In addition, if  $\text{gap} := \bar{z} - l$  is small, it is probable that  $R(l)$  is much smaller than the original in size. The reduced  $R(l)$  may be solved by some free or commercial IP solver, but in Section 5 we present a branch-and-bound algorithm to solve this problem, and the following algorithm solves BKP completely.

ALGORITHM VIRTUAL\_PEGGING

Step 1 Set  $l := \max\{\bar{z} - \alpha, \underline{z}\}$ .

Step 2 Carry out the pegging test with  $l$  and  $\bar{z}$  as lower and upper bounds, solve  $R(l)$  (by BRANCH\_AND\_BOUND given in Section 5) and obtain  $z_l^*$ .

Step 3 If  $l \leq z_l^*$ , go to Step 5.

Step 4 Update  $\underline{z} := \max\{\underline{z}, z_l^*\}$  and  $l := \max\{l - \alpha, \underline{z}\}$ , and go to Step 2.

Step 5 The optimal value is obtained as  $z^* = z_l^*$ .

Here,  $\alpha$  is an arbitrary small margin between the upper bound and the initially assumed lower bound. We set  $l := \bar{z} - \alpha$  at first if this is not smaller than  $\underline{z}$ . Then, if the optimal value is not found in Step 3,  $l$  is further lowered by  $\alpha$ , and we repeat Steps 2–4 all over again until an optimal solution is found.

## 5. Surrogate-based branch-and-bound

For two disjoint subsets  $F_0$  and  $F_1$  of  $\{1, 2, \dots, n\}$  we consider the subproblem (also referred to as a ‘node’) of BKP as

$P(F_0, F_1)$

$$\begin{aligned} & \text{maximize} && \min \left\{ \sum_{j=1}^n p_j^1 x_j, \sum_{j=1}^n p_j^2 x_j \right\} \\ & \text{subject to} && x \in X, \\ & && x_j = 0, \forall j \in F_0, \\ & && x_j = 1, \forall j \in F_1. \end{aligned}$$

Here  $F_0$  ( $F_1$ ) is the set of variables fixed at 0 (1, resp.), and by  $z^*(F_0, F_1)$  we denote the optimal objective value to this problem. Clearly  $P(\emptyset, \emptyset)$  is identical to BKP, and the problem  $P(F_0(l), F_1(l))$  is identical to  $R(l)$ .

Next, using  $\lambda^\dagger$  obtained by BINARY we define its relaxation as

$SP(F_0, F_1)$

$$\begin{aligned} & \text{maximize} && \sum_{j=1}^n \bar{p}_j x_j \\ & \text{subject to} && \sum_{j=1}^n w_j x_j \leq c, \\ & && x_j = 0, \quad \forall j \in F_0, \\ & && x_j = 1, \quad \forall j \in F_1, \\ & && 0 \leq x_j \leq 1, \quad \forall j \notin F_0 \cup F_1, \end{aligned}$$

and its optimal solution  $\bar{x}(F_0, F_1)$  with the corresponding objective value  $\bar{z}(F_0, F_1)$ . Note that  $SP(F_0, F_1)$  is a continuous knapsack problem which is easily solved. If these are infeasible, we define  $z^*(F_0, F_1) := -\infty$  and  $\bar{z}(F_0, F_1) := -\infty$ , respectively.

Then, a branch-and-bound algorithm [19] can be constructed as follows. We call BRANCH\_AND\_BOUND in Step 2 of the VIRTUAL\_PEGGING with  $F_0 := F_0(l)$  and  $F_1 := F_1(l)$ , and upon termination of this we obtain the optimal objective value as  $z_{\text{opt}}^* := z_l^*$ .

ALGORITHM BRANCH\_AND\_BOUND( $F_0, F_1$ )

*Step 1 (Initialization)* Let the incumbent optimal objective value be  $z_{\text{opt}}^* = -\infty$ .

*Step 2 (Evaluate the current subproblem)* Solve  $SP(F_0, F_1)$  and obtain  $\bar{x}(F_0, F_1)$  and  $\bar{z}(F_0, F_1)$ .

*Step 3 (Feasible solution)* If  $\bar{x}(F_0, F_1)$  is feasible to  $P(F_0, F_1)$ , go to Step 6.

*Step 4 (Inprospective node)* If  $\bar{z}(F_0, F_1) \leq z_{\text{opt}}^*$ , return.

*Step 5 (Branch and recursive call)* Do the following.

- (1) Find  $u := \min\{j \mid j \notin F_0 \cup F_1\}$ .
- (2) Call BRANCH\_AND\_BOUND( $F_0 \cup \{u\}, F_1$ ).
- (3) Call BRANCH\_AND\_BOUND( $F_0, F_1 \cup \{u\}$ ).
- (4) return.

*Step 6 (Update incumbent)* If  $z_{\text{opt}}^* < \bar{z}(F_0, F_1)$ , update  $z_{\text{opt}}^* := \bar{z}(F_0, F_1)$  and return.

By assumption **B**<sub>1</sub> and the definition of  $u$  above, the branching is made in the non-increasing order of  $\bar{p}_j/w_j$  among the unfixed variables. Also, by the recursive structure of the algorithm, subproblems are generated and examined in a depth-first fashion. The characteristic feature of

this algorithm is that the upper bound  $\bar{z}(F_0, F_1)$  can be computed quite rapidly, since  $SP(F_0, F_1)$  is a continuous knapsack problem.

### 6. A numerical example

Let us consider BKP with  $n = 10, c = 2500$  and the data of Table 1. Table 2 shows the binary search process, where we obtain  $\lambda_1^\dagger = 0.241, \bar{z} = 2652.75$  and  $\underline{z} = 2440$ . The gap between the bounds is 212.75. Table 3 shows the items in the non-increasing order of  $\bar{p}_j/w_j$ . Here the critical item is  $s = 6$ , and the thresholds are shown in the row of  $\theta_j$ . The row of  $x_j^*$  is the result of pegging, where ‘-’ indicates the unfixed variables.

Solving this BKP directly by calling `BRANCH_AND_BOUND( $\emptyset, \emptyset$ )`, we obtain the optimal  $z^* = 2440$  after examining 29 branch-and-bound nodes. If we apply the same method after reducing the problem by the pegging test (Theorem 1) with  $\bar{z} - \underline{z} = 212.75$ , the same solution is obtained after generating 15 subproblems. By the virtual pegging test with  $\alpha = 100$ , we get the solution after examining only 7 subproblems. However, in this case (4) of Theorem 2 is not satisfied since  $l = \bar{z} - \alpha = 2552.75 > z_j^* = 2440$ , and we need to run `VIRTUAL_PEGGING` again with  $l$  lowered to 2440 to get a guaranteed optimal solution. All these solutions coincide

Table 1. Data for the example of Section 6.

$j$	1	2	3	4	5	6	7	8	9	10
$w_j$	618	595	586	427	695	816	833	624	353	611
$p_j^1$	298	97	24	866	936	252	135	535	386	399
$p_j^2$	274	102	129	433	859	674	114	407	561	212

Table 2. Binary search process to find  $\lambda^\dagger$ .

Cycle	$\lambda_L$	$\lambda_R$	$\lambda_1$	$\bar{z}(\lambda)$	$\underline{z}(\lambda)$	$z^1 - z^2$
0	0.000	1.000	0.500	2719.03	2260	+
1	0.000	0.500	0.250	2655.12	2260	+
2	0.000	0.250	0.125	2657.80	2440	-
3	0.125	0.250	0.188	2655.04	2440	-
4	0.188	0.250	0.219	2653.67	2440	-
5	0.219	0.250	0.234	2652.98	2440	-
6	0.234	0.250	0.242	2653.12	2260	+
7	0.234	0.242	0.238	2652.80	2440	-
8	0.238	0.242	0.240	2652.72	2440	-
9	0.240	0.242	0.241	2652.88	2260	+
10	0.240	0.241	0.241	2652.75	2260	+

Table 3. Threshold  $\theta_j$  and pegging results.

$j$	9	5	4	8	6	1	10	3	2	7
$w_j$	353	695	427	624	816	618	611	586	595	833
$\bar{p}_j(\lambda^\dagger)$	518.9	877.5	537.2	437.8	572.4	279.8	257.0	103.7	100.8	119.8
$\theta_j$	271.2	390.0	237.7	0.1	0.0	-153.7	-171.6	-307.3	-316.6	-465.3
$x_j^*$	1	1	1	-	-	-	-	0	0	0

Downloaded By: [Yamada, Takeo] At: 02:45 12 June 2009

with those obtained from NUOPT [14], an MP/IP solver popular in Japan which is considered competitive to such solvers as LINDO, EXPRESS-MP, CPLEX, etc. [7].

## 7. Numerical experiments

### 7.1 Design of experiments

For BKP with  $n = 100$ – $16,000$  items, we evaluate the performance of the ‘surrogate relaxation + (virtual) pegging test + branch-and-bound’ approach developed in the previous sections. Weight  $w_j$  of item  $j$  is randomly and uniformly distributed over integer interval  $[1, 1000]$ , and the values of items are generated according to

- UNCOR (uncorrelated):  $p_j^k$  ( $k = 1, 2$ ) are distributed independently and uniformly over  $[1, 1000]$ ,
- WEAK (weakly correlated):  $p_j^k$  ( $k = 1, 2$ ) are distributed independently and uniformly over  $[w_j, w_j + 200]$ ,
- STRONG (strongly correlated):  $p_j^1 := w_j + 100$ , and  $p_j^2$  is distributed uniformly over  $[w_j, w_j + 200]$ .

Knapsack capacity is set to

$$c := 500n\rho$$

where  $\rho$  is either 0.25, 0.50 or 0.75. Since the average weight of an item is 500.5,  $\rho = 0.50$  means that approximately a half of all items can be accommodated in the knapsack.

To compute surrogate relaxation, pegging test and BRANCH\_AND\_BOUND, we have implemented the algorithm in ANSI C language on an IBM RS/6000 SP 44 Model 270 workstation (CPU: POWER 3-II, 375MHz). We also solved small instances using NUOPT Ver. 3.3.0 on the same machine.

### 7.2 Bounds and reduction

Tables 4 and 5 give the results of computation of the upper and lower bounds, as well as of the pegging test. Here, in addition to the bounds  $\bar{z}$  and  $\underline{z}$ , we show the gap  $(\bar{z} - \underline{z})$ , the relative error defined by

$$\text{rerror}(\%) = 100 \times \frac{\bar{z} - \underline{z}}{\underline{z}},$$

the number of unfixed variables ( $n'$ ), and the CPU time in seconds ( $\text{CPU}_0$ ) to evaluate the bounds and the effect of the pegging test. 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 very high precision in a few CPU seconds.
- (2) Gaps are smaller in WEAK case, but the reduced problems are of almost comparable size. In both of UNCOR and WEAK cases, problems are reduced considerably in size.
- (3)  $\text{CPU}_0$  increases with  $n$ , but this is rather insensitive to the correlation type of instances.
- (4) The objective value ( $\bar{z}$  and  $\underline{z}$ ) increases with  $\rho$ , but no significant influence of  $\rho$  is seen on the accuracy of heuristic solutions or CPU time.

Table 4. Upper and lower bounds and pegging result (UNCOR case).

$\rho$	$n$	$\bar{z}$	$\underline{z}$	gap	error (%)	$n'$	CPU <sub>0</sub>
0.25	2000	539,586.4	539,419.6	166.8	0.031	609.1	0.06
	4000	1,079,071.4	1,078,929.9	141.5	0.013	1043.2	0.16
	6000	1,624,944.1	1,624,749.8	194.3	0.012	2090.5	0.42
	8000	2,154,276.5	2,154,148.5	128.0	0.006	1848.7	0.52
	10,000	2,690,716.2	2,690,584.6	131.6	0.005	2414.9	0.64
	12,000	3,230,946.9	3,230,737.7	209.2	0.006	4397.7	1.53
	14,000	3,771,746.4	3,771,598.3	148.1	0.004	3714.2	1.40
	16,000	4,312,959.8	4,312,774.5	185.3	0.004	5194.8	2.39
0.50	2000	763,914.3	763,752.9	161.4	0.021	690.5	0.07
	4000	1,525,945.2	1,525,821.5	123.7	0.008	1101.2	0.26
	6000	2,290,905.1	2,290,762.2	142.9	0.006	1721.7	0.41
	8000	3,043,770.1	3,043,450.6	319.5	0.010	4697.2	1.47
	10,000	3,803,532.2	3,803,369.6	162.6	0.004	3591.2	1.32
	12,000	4,561,522.5	4,561,323.5	199.0	0.004	4671.0	2.29
	14,000	5,328,017.5	5,327,728.5	289.0	0.005	7073.0	4.26
	16,000	6,089,334.2	6,089,105.5	228.7	0.004	7195.4	4.00
0.75	2000	918,868.2	918,757.5	110.7	0.012	435.6	0.05
	4000	1,838,039.1	1,837,938.1	101.0	0.005	798.4	0.23
	6000	2,754,998.5	2,754,871.3	127.2	0.005	1412.7	0.50
	8000	3,665,178.5	3,665,075.7	102.8	0.003	1657.0	0.38
	10,000	4,579,119.1	4,579,022.7	96.4	0.002	1913.0	1.02
	12,000	5,490,787.8	5,490,581.2	206.6	0.004	4365.2	2.23
	14,000	6,413,209.0	6,413,037.1	171.9	0.003	3938.6	3.02
	16,000	7,326,788.9	7,326,515.2	273.7	0.004	6804.6	5.32

Table 5. Upper and lower bounds and pegging result (WEAK case).

$\rho$	$n$	$\bar{z}$	$\underline{z}$	gap	error (%)	$n'$	CPU <sub>0</sub>
0.25	2000	357,159.9	357,120.3	39.6	0.001	720.7	0.08
	4000	714,453.0	714,417.4	35.6	0.005	1204.7	0.21
	6000	1,072,339.1	1,072,307.9	31.2	0.003	1724.4	0.51
	8000	1,429,168.4	1,429,140.6	27.8	0.002	2089.0	0.57
	10,000	1,786,782.7	1,786,748.8	33.9	0.002	3078.3	0.90
	12,000	2,143,235.6	2,143,196.7	38.9	0.002	4288.6	1.71
	14,000	2,501,692.2	2,501,650.9	41.3	0.002	5220.0	2.42
	16,000	2,859,882.9	2,859,830.3	52.6	0.002	7239.7	3.82
0.50	2000	651,452.4	651,415.3	37.1	0.006	805.4	0.10
	4000	1,302,535.2	1,302,488.6	46.6	0.004	1918.8	0.39
	6000	1,954,819.6	1,954,775.3	44.3	0.002	2667.3	0.58
	8000	2,606,223.9	2,606,167.5	56.4	0.002	4530.3	1.44
	10,000	3,257,987.2	3,257,950.2	37.0	0.001	4064.5	1.84
	12,000	3,908,854.0	3,908,793.7	60.3	0.002	7088.5	3.34
	14,000	4,561,511.9	4,561,454.2	57.7	0.001	7603.3	4.82
	16,000	5,213,789.2	5,213,744.0	45.2	0.001	7604.7	4.99
0.75	2000	932,043.5	931,997.9	45.6	0.005	849.1	0.14
	4000	1,863,882.4	1,863,840.3	42.1	0.002	1648.4	0.44
	6000	2,796,998.2	2,796,951.7	46.5	0.002	2555.2	0.73
	8000	3,729,220.2	3,729,178.1	42.1	0.001	3199.7	1.07
	10,000	4,662,129.1	4,662,087.8	41.3	0.001	3790.4	2.34
	12,000	5,593,840.7	5,593,811.9	28.8	0.001	3472.7	3.00
	14,000	6,527,118.2	6,527,070.5	47.7	0.001	6112.9	3.91
	16,000	7,460,437.6	7,460,387.5	50.1	0.001	7433.5	5.87

### 7.3 Exact solution

To solve BKP exactly, we compare the following three methods.

- PEG-NUOPT: Apply the pegging test (Theorem 1) to BKP, and solve the reduced problem using NUOPT.
- PEG-BAB: Apply the same pegging test, and then solve the reduced problem by calling BRANCH\_AND\_BOUND.
- VPEG-BAB: Solve BKP by calling VIRTUAL\_PEGGING with  $l := \bar{z} - \alpha$ .

Tables 6 and 7 compare PEG-NUOPT against PEG-BAB. For each value of  $\rho$  and  $n$ , we computed the randomly generated 10 instances, with the computation truncated at the time limit of TL=1800 seconds. The column of ‘#sol’ show the number of runs completed within TL, and all the rows are the average over the finished instances. In all solved cases, we obtained the identical objective values. In these tables, ‘BBN’ is the number (in millions) of the branch-and-bound nodes generated, and ‘CPU’ is the time in seconds to solve the problem completely by respective methods.

The observation from these tables are:

- (1) PEG-NUOPT was able to solve problems of  $n = 2000$  or smaller, but for larger  $n$  it often fails; especially for  $n \geq 10,000$  this approach is almost hopeless irrespective to  $\rho$  and correlation type of instances.
- (2) PEG-BAB solved all problems with  $n \leq 6000$ , and frequently solved larger problems. No clear difference was seen in the difficulty of solving UNCOR and WEAK instances.

Table 6. Exact solution (UNCOR case).

$\rho$	$n$	PEG-NUOPT		PEG-BAB		
		CPU	#sol	BBN ( $\times 10^6$ )	CPU	#sol
0.25	2000	90.1	10	3.02	1.31	10
	4000	383.5	10	11.98	5.03	10
	6000	478.6	4	68.69	28.33	10
	8000	608.2	3	22.48	9.72	10
	10,000	–	0	318.88	129.68	9
	12,000	–	0	60.19	25.49	6
	14,000	1129.2	1	142.23	59.02	9
	16,000	1100.5	2	618.84	254.49	9
0.50	2000	148.1	10	4.01	1.73	10
	4000	630.4	5	19.47	8.21	10
	6000	752.7	4	200.81	81.57	10
	8000	828.1	1	105.55	45.00	8
	10,000	488.3	1	165.03	68.55	10
	12,000	871.9	1	529.60	218.42	8
	14,000	–	0	38.48	16.18	3
	16,000	–	0	101.27	229.45	8
0.75	2000	93.3	10	2.79	1.22	10
	4000	418.8	8	6.79	3.01	10
	6000	225.8	4	13.69	6.21	10
	8000	705.1	6	64.44	26.50	8
	10,000	535.8	2	32.57	14.18	9
	12,000	1119.1	1	65.21	29.17	7
	14,000	–	0	29.55	13.60	8
	16,000	1078.4	1	61.65	28.15	5

Downloaded By: [Yamada, Takeo] At: 02:45 12 June 2009

Table 7. Exact solution (WEAK case).

$\rho$	$n$	PEG–NUOPT		PEG–BAB		
		CPU	#sol	BBN ( $\times 10^6$ )	CPU	#sol
0.25	2000	222.3	10	5.48	2.31	10
	4000	616.5	6	7.84	3.43	10
	6000	725.5	2	61.53	25.50	10
	8000	1537.9	2	18.49	8.13	10
	10,000	–	0	19.73	8.83	8
	12,000	–	0	201.81	84.09	10
	14,000	934.0	1	23.67	12.53	10
	16000	–	0	127.45	309.85	7
0.50	2000	545.9	10	5.87	2.51	10
	4000	693.5	3	24.90	10.60	10
	6000	761.4	3	108.03	44.83	10
	8000	694.9	1	17.81	9.46	7
	10,000	–	0	36.42	17.06	9
	12,000	–	0	591.22	244.76	8
	14,000	–	0	727.68	303.20	9
	16,000	–	0	110.93	51.25	8
0.75	2000	399.7	10	7.07	3.05	10
	4000	978.8	4	13.89	6.15	10
	6000	218.4	1	226.01	92.21	10
	8000	1574.7	1	24.62	11.13	9
	10,000	998.6	3	25.32	13.17	8
	12,000	–	0	23.77	12.85	8
	14,000	1208.0	2	171.23	74.40	9
	16,000	–	0	154.53	70.37	8

### 7.4 Virtual pegging result

VPEG-BAB was run with the initial margin

$$\alpha := 1000 \cdot \log^2 \frac{n}{n} \tag{20}$$

This follows Lueker [21], who proved for the 0-1 knapsack problem (KP)

$$\bar{z}_{KP} - z_{KP}^* = O\left(\log^2 \frac{n}{n}\right)$$

where  $\bar{z}_{KP}$  and  $z_{KP}^*$  are, respectively, the continuously relaxed upper bound and the optimal objective value. After some preliminary experiments, we set the constant in Equation (20) as 1000. In all our computation with Equation (20),

$$\bar{z} - \alpha < z^*$$

was confirmed *a posteriori*, and thus  $\bar{z} - \alpha$  was indeed a correct lower bound in all instances tested. Therefore, in VIRTUAL\_PEGGING Steps 2 through 4 were actually repeated only once. The results are shown in Tables 8 and 9, where in addition to BBN, CPU and #sol, the number of unfixed variables by virtual pegging ( $n''$ ), and the optimal objective value ( $z^*$ ) are shown.

Except for a few correlated instances, almost all problems were solved within a few minutes. Comparing Tables 6 and 8 (also 7 and 9), the effectiveness of the virtual pegging approach is evident.

Table 8. Exact solution by VPEG-BAB (UNCOR case).

$\rho$	$n$	$\alpha$	$n''$	$z^*$	BBN( $\times 10^6$ )	CPU	#sol
0.25	2000	28.89	115.7	539,572.2	1.24	0.55	10
	4000	17.20	131.8	1,079,062.3	2.89	1.27	10
	6000	12.61	137.5	1,624,937.1	10.64	4.48	10
	8000	10.10	148.9	2,154,272.2	5.04	2.22	10
	10,000	8.48	158.3	2,690,712.3	40.07	16.45	10
	12,000	7.35	167.1	3,230,943.6	38.94	16.03	10
	14,000	6.51	173.7	3,771,743.7	17.11	7.26	10
	16,000	5.86	174.0	4,312,957.7	17.58	7.49	10
0.50	2000	28.89	132.7	763,902.3	1.72	0.75	10
	4000	17.20	167.1	1,525,938.1	7.52	3.13	10
	6000	12.61	183.6	2,290,900.8	12.04	5.01	10
	8000	10.10	192.6	3,043,766.6	17.92	7.43	10
	10,000	8.48	207.2	3,803,529.5	14.19	5.94	10
	12,000	7.35	211.0	4,561,519.8	25.63	10.62	10
	14,000	6.51	218.4	5,328,015.1	391.06	160.32	10
	16,000	5.86	229.6	6,089,332.0	44.27	18.24	10
0.75	2000	28.89	118.7	918,854.7	1.64	0.72	10
	4000	17.20	140.9	1,838,031.7	2.49	1.09	10
	6000	12.61	159.4	2,754,994.0	5.72	2.46	10
	8000	10.10	170.1	3,665,174.7	35.04	14.46	10
	10,000	8.48	170.1	4,579,116.2	19.10	7.94	10
	12,000	7.35	177.2	5,490,784.4	65.82	27.89	10
	14,000	6.51	185.5	6,413,206.3	48.47	19.99	10
	16,000	5.86	190.8	7,326,786.7	189.54	81.75	10

Table 9. Exact solution by VPEG-BAB (WEAK case).

$\rho$	$n$	$\alpha$	$n''$	$z^*$	BBN( $\times 10^6$ )	CPU	#sol
0.25	2000	28.89	500.0	357,157.3	4.55	1.85	10
	4000	17.20	602.4	714,451.8	5.15	2.13	10
	6000	12.61	694.0	1,072,338.4	14.40	5.86	10
	8000	10.10	755.7	1,429,167.8	7.28	3.07	10
	10,000	8.48	788.1	1,786,782.0	91.46	37.46	10
	12,000	7.35	821.2	2,143,235.3	10.64	4.51	10
	14,000	6.51	836.3	2,501,691.6	6.29	2.83	10
	16,000	5.86	879.8	2,859,882.3	160.19	238.81	10
0.50	2000	28.89	629.0	651,449.6	5.43	2.21	10
	4000	17.20	811.8	1,302,534.1	8.81	3.58	10
	6000	12.61	890.8	1,954,818.7	12.01	4.91	10
	8000	10.10	940.7	2,606,223.1	588.09	377.09	10
	10,000	8.48	991.2	3,257,986.8	10.20	4.27	10
	12,000	7.35	1020.6	3,908,853.7	160.21	64.09	10
	14,000	6.51	1058.3	4,561,511.8	157.68	62.61	10
	16,000	5.86	1110.8	5,214,818.3	41.33	16.70	8
0.75	2000	28.89	564.1	932,040.4	6.95	2.82	10
	4000	17.20	699.5	1,863,881.2	9.39	3.82	10
	6000	12.61	738.7	2,796,997.4	140.28	56.95	10
	8000	10.10	810.2	3,729,219.5	29.56	11.99	10
	10,000	8.48	843.6	4,662,087.1	161.29	66.85	9
	12,000	7.35	891.3	5,594,031.2	376.50	149.46	9
	14,000	6.51	918.3	6,527,118.0	72.05	28.84	10
	16,000	5.86	930.2	7,460,150.6	30.00	12.29	9

### 7.5 Comparison against other commercial solvers

We further evaluate PEG-BAB against two leading IP solvers CPLEX 10.1 [11] and XPRESS-MP release 2005B [8] on an even faster computer DELL Dimension 8400 (Pentium(R) 4 CPU, 3.40 GHz). Table 10 gives the summary of CPU times in solving UNCOR and WEAK instances with  $\rho = 0.25$  and  $n = 2000, 4000, 8000$ . Ten instances solved in each row are identical to those in the corresponding rows of Tables 6 and 7. Here the columns show the CPU time in seconds of the following solution methods:

- CPLEX: Solve BKP<sup>d</sup> directly by using CPLEX.
- PEG-CPLEX: Apply the pegging test, and then solve the reduced problem by CPLEX.
- PEG-XPRESS: Also, solve the reduced problem by XPRESS-MP.

We observe the pegging test effective as a pre-processing in solving the problem, and PEG-BAB is substantially advantageous to these commercial solvers.

### 7.6 Small and strongly correlated instances

In standard 0–1 knapsack problems, it is known that small instances are often harder to solve to optimality than the larger ones [6]. Table 11 summarizes the result of experiments for BKP with  $n$  between 100 and 500, where we compare PEG-CPLEX and VPEG-BAB. As in previous tables, each row is the average over the solved cases out of 10 randomly generated instances, and  $n''$  is the number of variables in the reduced problem.

Table 10. CPU seconds of IP solvers vs. PEG-BAB for instances with  $\rho = 0.25$ .

Problem	$n$	CPLEX	PEG-CPLEX	PEG-XPRESS	PEG-BAB
UNCOR	2000	15.46	6.01	9.64	0.41
	4000	55.33	16.29	21.61	1.57
	8000	266.31	81.59	87.33	3.11
WEAK	2000	49.11	21.21	21.13	0.75
	4000	66.83	22.55	31.86	1.19
	8000	147.53	62.32	167.18	10.80

Table 11. Small size instances.

Problem	$n$	PEG-CPLEX		VPEG-BAB		
		CPU	#sol	$n''$	CPU	#sol
UNCOR	100	0.06	10	31.8	0.00	10
	200	0.12	10	43.9	0.00	10
	300	0.40	10	58.7	0.01	10
	400	0.81	10	65.3	0.01	10
	500	0.56	10	71.0	0.01	10
WEAK	100	0.16	10	81.9	0.00	10
	200	0.57	10	149.8	0.01	10
	300	1.60	10	191.6	0.05	10
	400	1.23	10	171.3	0.06	10
	500	2.17	10	317.0	0.06	10
STRONG	100	0.22	10	64.0	0.75	10
	200	19.40	9	150.9	18.82	8
	300	10.93	9	239.1	41.76	6
	400	1.16	9	313.9	22.82	5
	500	1.37	6	384.5	101.48	1

From this, we observe that UNCOR and WEAK instances of this size are easily solved by both CPLEX and VPEG-BAB. On the other hand, in STRONG we often encounter instances that can not be solved within the fixed time limit of 1800 seconds, while some instances are solved within a few seconds. Especially, in STRONG case the virtual pegging method becomes less effective as  $n$  increases, and this makes VPEG-BAB unsatisfactory in such a case.

## 8. Conclusion

We gave heuristic and exact algorithms to solve BKP. In addition to the surrogate relaxation, the pegging test and the surrogate-based branch-and-bound method, we presented a virtual pegging algorithm. For problems with  $n \leq 16,000$  we were able to obtain approximate solutions of quite high accuracy in less than a few seconds, and except for some instances, exact solutions were obtained by VPEG-BAB in less than a few minutes on an ordinary computer. However, strongly correlated instances were difficult to solve by this method, unless the problem size was very small.

## Acknowledgement

The authors are grateful to anonymous referees for their careful reading of the manuscript and helpful comments.

## References

- [1] E. Balas and E. Zemel, *An algorithm for large zero-one knapsack problems*, Oper. Res. 28 (1980), pp. 1130–1154.
- [2] CPLEX 10.0, ILOG, 2007; software available at <http://www.ilog.com/products/cplex/news/whatsnew.cfm>.
- [3] R.S. Dembo and P.L. Hammer, *A reduction algorithm for knapsack problems*, Methods Oper. Res. 36 (1980), pp. 49–60.
- [4] M. Eben-Chaime, *Parametric solution for linear bicriteria knapsack models*, Manag. Sci. 42 (1996), pp. 1565–1575.
- [5] D. Fayard and G. Plateau, *Resolution of the 0-1 knapsack problem: comparison of methods*, Math. Program. 8 (1975), pp. 272–307.
- [6] M. Fisher, *The Lagrangian relaxation method for solving integer programming problems*, Manage. Sci. 50 (2004), pp. 1861–1871.
- [7] R. Fourer, *Software survey: linear programming*, OR/MS Today 26 (1999), pp. 64–71.
- [8] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman and Company, New York, 1979.
- [9] F. Glover, *Surrogate constraint duality in mathematical programming*, Oper. Res. 23 (1975), pp. 434–451.
- [10] H. Iida, *A note on the max-min 0-1 knapsack problem*, J. Comb. Optim. 3 (1999), pp. 89–94.
- [11] G.P. Ingargiola and F.F. Korsh, *Reduction algorithms for zero-one single knapsack problems*, Manage. Sci. 20 (1973), pp. 460–463.
- [12] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*, Springer, Berlin, 2004.
- [13] P. Kouvelis and G. Yu, *Robust Discrete Optimization and Its Applications*, Kluwer, Dordrecht, 1997.
- [14] G. Lueker, *On the average difference between the solutions to linear and integer knapsack problems*, in *Applied Probability – Computer Science, The Interface I*, R.L. Disney and T.J. Otter, eds., Birkhauser, Boston, MA, 1982, pp. 489–504.
- [15] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*, Wiley, New York, 1990.
- [16] NUOPT Ver. 3.3.0, 2002; software available at <http://www.msi.co.jp/nuopt>.
- [17] F. Taniguchi, T. Yamada, and S. Kataoka, *Heuristic and exact algorithms for the max-min optimization of the multi-criteria knapsack problem*, Comput. Oper. Res. 35 (2008), pp. 2034–2048.
- [18] L.A. Wolsey, *Integer Programming*, Wiley, New York, 1998.
- [19] XPRESS-MP release 2005B, Dash Optimization, 2005; software available at <http://www.dashoptimization.com>.
- [20] B.-J. You and T. Yamada, *A virtual pegging approach to the precedence constrained knapsack problem*, Eur. J. Oper. Res. 183 (2007), pp. 618–632.
- [21] G. Yu, *On the max-min 0-1 knapsack problem with robust optimization applications*, Oper. Res. 44 (1996), pp. 407–415.
- [22] C.-N. Zhang and H.-L. Ong, *Solving the bicriteria zero-one knapsack problem by an efficient LP-based heuristic*, Eur. J. Oper. Res. 159 (2004), pp. 545–557.