

Manuscript Number: EJOR-D-09-01546

Title: An exact algorithm for the budget-constrained multiple knapsack problem

Article Type: Theory and Methodology Paper

Section/Category: Integer programming

Keywords: Multiple knapsack problem; Combinatorial optimization; Integer programming.

Corresponding Author: doctoral student Byungjun You, M.S.

Corresponding Author's Institution: National Defense Academy of Japan

First Author: Byungjun You, M.S.

Order of Authors: Byungjun You, M.S.; Takeo Yamada, Ph. D.

Abstract: We formulate the budget-constrained multiple knapsack problem (BCMKP) as an extension of the multiple knapsack problem (MKP). The Lagrangian relaxation problem is easily solved, and together with a greedy heuristic we obtain a pair of upper and lower bounds quickly. We make use of these bounds in the pegging test to reduce the problem size. We also present a branch-and-bound algorithm to solve BCMKP to optimality. This algorithm makes use of the Lagrangian multipliers for pruning, and at each terminal subproblem solve MKP exactly by calling the MULKNAP code developed by Prof. D. Pisinger. As a result, we are able to solve test problems with up to 160000 items and 150 knapsacks within a few minutes in an ordinary computing environment, although the algorithm remains some weakness for small instances with relatively many knapsacks.

An exact algorithm for the budget-constrained multiple knapsack problem

Byungjun You* and Takeo Yamada†

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

Abstract

We formulate the budget-constrained multiple knapsack problem (BCMKP) as an extension of the multiple knapsack problem (MKP). The Lagrangian relaxation problem is easily solved, and together with a greedy heuristic we obtain a pair of upper and lower bounds quickly. We make use of these bounds in the pegging test to reduce the problem size. We also present a branch-and-bound algorithm to solve BCMKP to optimality. This algorithm makes use of the Lagrangian multipliers for pruning, and at each terminal subproblem solve MKP exactly by calling the MULKNAP code developed by Prof. D. Pisinger. As a result, we are able to solve test problems with up to 160000 items and 150 knapsacks within a few minutes in an ordinary computing environment, although the algorithm remains some weakness for small instances with relatively many knapsacks.

Keywords: Multiple knapsack problem; Combinatorial optimization, Integer programming.

1 Introduction

This paper is concerned with a variation of the *multiple knapsack problem* (MKP) [6, 9, 10], where we are given a set of n items $N = \{1, 2, \dots, n\}$ to be packed into m possible knapsacks $M = \{1, 2, \dots, m\}$. As in ordinary MKP, by w_j and p_j we denote the weight and profit of item $j \in N$ respectively, and the capacity of knapsack $i \in M$ is c_i . However, a fixed cost f_i is imposed if we use knapsack i , and knapsacks are freely available within a fixed budget B . The problem is to determine the set of knapsacks to be employed as well as to fill the adopted knapsacks with items such that the capacity constraints are all satisfied and the total profit is maximized. Let y_i and x_{ij} be the decision variables such that $y_i = 1$ if we use knapsack i , and $y_i = 0$ otherwise. Also, $x_{ij} = 1$ if item j is put into knapsack i , and $x_{ij} = 0$ otherwise. Then, the problem is formulated as the following *budget-constrained multiple knapsack problem*.

*Corresponding author. E-mail: g48095@nda.ac.jp

†E-mail: yamada@nda.ac.jp

BCMKP:

$$\text{maximize} \quad \sum_{i=1}^m \sum_{j=1}^n p_j x_{ij} \quad (1)$$

$$\text{subject to} \quad \sum_{j=1}^n w_j x_{ij} \leq c_i y_i, \quad i \in M, \quad (2)$$

$$\sum_{i=1}^m x_{ij} \leq 1, \quad j \in N, \quad (3)$$

$$\sum_{i=1}^m f_i y_i \leq B, \quad (4)$$

$$x_{ij}, y_i \in \{0, 1\}, \quad i \in M, j \in N. \quad (5)$$

Throughout the paper we assume the following.

A₁. Problem data w_j, p_j ($j \in N$), c_i, f_i ($i \in M$) and B are all positive integers.

A₂. Items are arranged in non-increasing order of profit per weight, i.e.,

$$p_1/w_1 \geq p_2/w_2 \geq \dots \geq p_n/w_n. \quad (6)$$

A₃. Knapsacks are numbered in non-increasing order of capacity per cost, i.e.,

$$c_1/f_1 \geq c_2/f_2 \geq \dots \geq c_m/f_m. \quad (7)$$

BCMKP is \mathcal{NP} -hard [4], since the special case of free knapsacks ($f_i \equiv 0, \forall i \in M$) is simply an MKP which is already \mathcal{NP} -hard. Since BCMKP is a linear 0-1 programming problem, small instances can be solved using MIP (mixed integer programming) solvers. In this article, we present an algorithm that solves larger BCMKPs to optimality within a few minutes in an ordinary computing environment.

This problem is closely related to the *fixed-charge multiple knapsack problem* (FCMKP, [16]), where instead of the budget constraint (4) the objective is to maximize the *net* total profit $\sum_{i=1}^m \sum_{j=1}^n p_j x_{ij} - \sum_{i=1}^m f_i y_i$. Both of these problems may be encountered by a shipping company in making up a ship leasing plan to transport items from one location to the other.

2 Upper and lower bounds

This section derives an *upper bound* by applying the *Lagrangian relaxation* [3] to BCMKP. We also present a *greedy* algorithm to obtain a good approximate solution quickly, which necessarily gives a *lower bound* to BCMKP.

2.1 Lagrangian relaxation

With nonnegative multipliers $\lambda = (\lambda_i) \in R^m$ and $\mu \in R$ associated with (2) and (4) respectively, the *Lagrangian relaxation* to BCMKP is

1 **LBCMKP**(λ, μ):

$$2 \quad \text{maximize} \quad \sum_{i=1}^m \sum_{j=1}^n (p_j - \lambda_i w_j) x_{ij} + \sum_{i=1}^m (\lambda_i c_i - \mu f_i) y_i + \mu B \quad (8)$$

$$3 \quad \text{subject to} \quad (3), (5).$$

4
5
6
7 For $\lambda \geq \mathbf{0}, \mu \geq 0$, let $\bar{z}(\lambda, \mu)$ denote the optimal objective value to LBCMKP(λ, μ). Then, it is
8 easily proved that $\bar{z}(\lambda, \mu)$ gives an upper bound to BCMKP, and $\bar{z}(\lambda, \mu)$ is a piecewise linear
9 and convex function of λ and μ . Moreover, if we consider the *Lagrangian dual*

$$10 \quad \text{minimize} \quad \bar{z}(\lambda, \mu) \quad \text{subject to} \quad \lambda \geq \mathbf{0}, \mu \geq 0,$$

11 we have the following.

12
13
14
15 **Theorem 1** *There exists an optimal solution $\lambda^\dagger = (\lambda_i^\dagger)$ to the Lagrangian dual such that*
16 $\lambda_1^\dagger = \lambda_2^\dagger = \dots = \lambda_m^\dagger (\equiv \lambda^\dagger)$.

17
18
19 **Proof.** For a fixed $\lambda = (\lambda_i) \geq \mathbf{0}$, let $k := \arg \min_{i \in M} \{\lambda_i\}$. Then, since $p_j - \lambda_i w_j \leq p_j - \lambda_k w_j$ for
20 all $i \in M$ and $j \in N$, the objective function (8) is maximized by setting $x_{ij} = 1$ if and only if
21 $i = k$ and $p_j - \lambda_k w_j > 0$. Similarly, $y_i = 1$ if and only if $\lambda_i c_i - \mu f_i > 0$, and we obtain

$$22 \quad \bar{z}(\lambda, \mu) = \sum_{j=1}^n (p_j - \lambda_k w_j)^+ + \sum_{i=1}^m (\lambda_i c_i - \mu f_i)^+ + \mu B, \quad (9)$$

23
24
25
26
27
28 where $(\cdot)^+ := \max\{\cdot, 0\}$. Here, we note that $(\lambda_i c_i - \mu f_i)^+$ is monotonically non-decreasing with
29 respect to λ_i ($i \neq k$). Thus, under the condition $\lambda_i \geq \lambda_k$, (9) is minimized at $\lambda_i \equiv \lambda_k$ ($i \in M$). ■

30
31
32 From this theorem, it suffices to consider the case of $\lambda_i \equiv \lambda$ ($\forall i \in M$), and thus $\bar{z}(\lambda, \mu)$ is
33 rewritten as

$$34 \quad \bar{z}(\lambda, \mu) = (C_t - W_s)\lambda + (B - F_t)\mu + P_s. \quad (10)$$

35
36
37 Here s and t are the *critical values* defined as $s := \max\{j \mid p_j - \lambda w_j \geq 0\}$ and $t := \max\{i \mid$
38 $\lambda c_i - \mu f_i \geq 0\}$ respectively, and W_s is the *accumulated weight* given by $W_s := \sum_{j=1}^s w_j$. $P_s,$
39 C_t and F_t are analogously defined for $(p_j), (c_i)$ and (f_i) , respectively. For a fixed $\mu \geq 0$, (10)
40 is a piecewise linear function of λ (See Figure 1), and its gradient changes from $C_t - W_s$ to
41 $C_t - W_{s-1}$ as λ increases from $p_s/w_s - 0$ to $p_s/w_s + 0$. Similarly, the gradient increases from
42 $C_{t-1} - W_s$ to $C_t - W_s$ at $\lambda = (f_t/c_t)\mu$. Thus, we obtain the optimal λ as

$$43 \quad \lambda^\dagger(\mu) = \begin{cases} p_s/w_s, & \text{if } W_{s-1} \leq C_t \leq W_s, \\ (f_t/c_t)\mu, & \text{if } C_{t-1} \leq W_s \leq C_t. \end{cases} \quad (11)$$

44
45
46
47
48
49
50 Putting this into (10), $\bar{z}(\lambda^\dagger(\mu), \mu)$ is also piecewise linear with respect to μ , and by *bisection*
51 *method* [14] we obtain an optimal μ^\dagger and $\lambda^\dagger := \lambda^\dagger(\mu^\dagger)$, and correspondingly an upper bound
52 $\bar{z} = \bar{z}(\lambda^\dagger(\mu^\dagger), \mu^\dagger)$. Let us introduce the *thresholds* as

$$53 \quad \theta_j := p_j - \lambda^\dagger w_j, \quad \eta_i := \lambda^\dagger c_i - \mu^\dagger f_i. \quad (12)$$

54
55
56 Then, from (9) the Lagrangian upper bound is given as

$$57 \quad \bar{z} := \bar{z}(\lambda^\dagger, \mu^\dagger) = \sum_{j \in N} \theta_j^+ + \sum_{i \in M} \eta_i^+ + \mu^\dagger B. \quad (13)$$

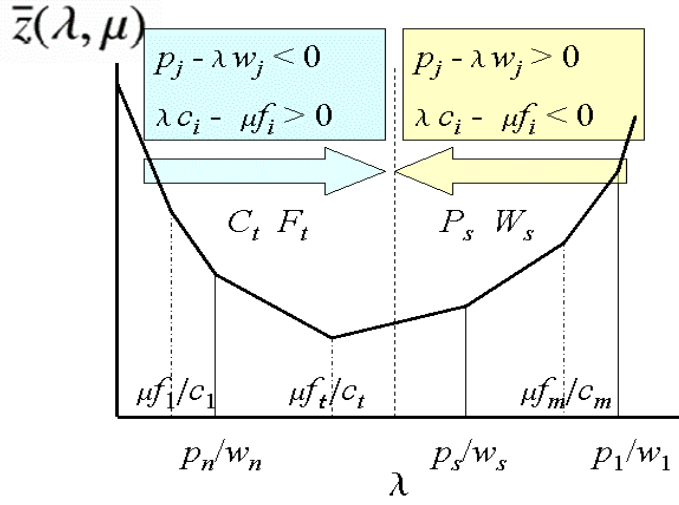


Figure 1: Lagrangian function $\bar{z}(\lambda, \mu)$ for a fixed μ .

Remark 1. We note that constraint (5) in $\text{LBCMKP}(\lambda, \mu)$ can be replaced with

$$x_{ij} \geq 0, \quad 0 \leq y_i \leq 1, \quad i \in M, j \in N. \quad (14)$$

Then, the Lagrangian upper bound \bar{z} coincides with the upper bound \bar{z}_C obtained by the continuous relaxation of BCMKP (see, e.g., Wolsey [15], Theorem 10.3 and Corollary).

Example 1. We consider the following instance with budget $B = 1627$.

Table 1: The data of items

j	1	2	3	4	5	6	7	8	9	10	11	12
p_j	621	466	456	645	756	609	923	302	613	670	456	427
w_j	62	66	129	255	410	552	873	293	687	802	591	571
p_j/w_j	10.02	7.06	3.53	2.53	1.84	1.10	1.06	1.03	0.89	0.84	0.77	0.75

Table 2: The data of knapsacks

i	1	2	3	4	5
c_i	953	326	788	126	804
f_i	546	223	788	156	1000
c_i/f_i	1.75	1.46	1.00	0.81	0.80

By applying the Lagrangian relaxation we obtain the upper bound $\bar{z}_L = 4239.744$ with $\lambda^\dagger = 1.057$ and $\mu^\dagger = 0.854$. On the other hand, solving the continuous relaxation of this problem with CPLEX 11.1 we have $\bar{z}_C = 4239.740$, and the dual variables $\lambda_i = 1.057$ for (2) and $\mu = 0.854$ for (4). ■

2.2 A lower bound

A lower bound to BCMKP can be obtained as follows. First, we determine the set of knapsacks to be used by solving the following knapsack problem for knapsacks.

$$\begin{aligned} & \text{maximize} && \sum_{i=1}^m c_i y_i \\ & \text{subject to} && \sum_{i=1}^m f_i y_i \leq B, \\ & && y_i \in \{0, 1\}, i \in M. \end{aligned}$$

This can be solved exactly or approximately using algorithms found in, e.g., Horowitz-Sahni [5] or Pisinger [12]. Once (y_i) is fixed, we can solve the resulting MKP using MULKNAP code due to Pisinger [13], and obtain a lower bound \underline{z} .

Example 2. For the instance of Example 1, we solve the above knapsack problem, and obtain the optimal solution $\underline{y}_1 = \underline{y}_2 = \underline{y}_3 = 1$ with the lower bound $\underline{z} = 3867$. ■

3 Problem reduction

A pegging test is well known for general 0-1 programming problem [2, 8, 11]. By applying this test, many variables are fixed either at 0 or 1, and removing these we obtain a problem of (often significantly) reduced size. This method can be applied to BCMKP, but to do so we need to solve the continuous relaxation of BCMKP, and this is often time-consuming for large-scale instances. We introduce a novel pegging test that makes use of the Lagrangian multipliers and easy to compute. A generalization of this pegging test can also be exploited later in developing a branch-and-bound algorithm to solve BCMKP to optimality.

Assume that we have the optimal Lagrangian multipliers λ^\dagger and μ^\dagger with the corresponding upper bound \bar{z} given by (13), as well as the lower bound \underline{z} obtained in Section 2.2. Let δ be either 0 or 1, and we introduce $P(y_k = \delta)$ as the subproblem of BCMKP with y_k fixed at δ . $\bar{P}(y_k = \delta)$ denotes the Lagrangian relaxation of $P(y_k = \delta)$ using the optimal λ^\dagger and μ^\dagger in (8). That is,

$\bar{P}(y_k = \delta)$:

$$\begin{aligned} & \text{maximize} && \sum_{j=1}^n \theta_j x_j + \sum_{i=1}^m \eta_i y_i + \mu^\dagger B \end{aligned} \tag{15}$$

$$\text{subject to} \quad x_j, y_i \in \{0, 1\}, \forall j \in N, \forall i \in M, y_k = \delta, \tag{16}$$

where we introduce a new 0-1 variable x_j defined by $x_j = \sum_{i=1}^m x_{ij}$.

Let $(\mathbf{x}^*, \mathbf{y}^*)$ be an optimal solution to BCMKP with $\mathbf{x}^* = (x_{ij}^*)$ and $\mathbf{y}^* = (y_i^*)$. Then, the following is immediate.

Theorem 2 (Pegging of knapsacks) For every $k \in M$,

(i) If $\eta_k > 0$ and $\bar{z} - \underline{z} \leq \eta_k \Rightarrow y_k^* = 1$,

(ii) If $\eta_k < 0$ and $\bar{z} - \underline{z} \leq -\eta_k \Rightarrow y_k^* = 0$.

Proof. (i) Note that the optimal objective value to $\bar{P}(y_k = \delta)$ is $\bar{z}(y_k = \delta) := \sum_{j \in N} \theta_j^+ + \sum_{i \in M, i \neq k} \eta_i^+ + \mu^\dagger B + \eta_k \delta$. Then, comparing this with (13) we have $\bar{z}(y_k = 0) = \bar{z} - \eta_k \leq \underline{z}$, which implies $y_k^* = 1$ in any optimal solution. (ii) is similarly proved. ■

Applying Theorem 2, the knapsacks are classified into three disjoint subsets $K_0 := \{i \in M | y_i^* = 0\}$, $K_1 := \{i \in M | y_i^* = 1\}$ and the remaining $M \setminus (K_0 \cup K_1)$. Knapsack $i \in K_0$ is never used, while $i \in K_1$ is always used in any optimal solution to BCMKP.

Similarly, we can derive a pegging theorem for items, and applying this classify items into the disjoint sets $I_0 := \{j \in N | x_j^* = 0\}$, $I_1 := \{j \in N | x_j^* = 1\}$ and $N \setminus (I_0 \cup I_1)$. Removing K_0 and I_0 , BCMKP is reduced (often significantly) in size. In what follows, we assume that these are already done, and thus $K_0 = I_0 = \emptyset$.

Example 3. For the instance of Example 1, the optimal multipliers are $\lambda^\dagger = 1.057$ and $\mu^\dagger = 0.854$ and the thresholds are as follows.

Table 3: The threshold θ_j for items.

j	1	2	3	4	5	6	7	8	9	10	11	12
θ_j	555.5	396.2	319.6	375.4	322.5	25.4	0.0	-7.8	-113.4	-177.9	-168.9	-176.7

Table 4: The threshold η_i for knapsacks.

i	1	2	3	4	5
η_i	541.3	154.2	160.2	0.0	-4.0

With the $gap := \bar{z} - \underline{z} = 372.7$, we fix $x_1^* = x_2^* = x_4^* = 1$ and $y_1^* = 1$. ■

4 A branch-and-bound algorithm

A characteristic feature of the branch-and-bound algorithm to be given below is that branches are made with respect to variables (y_i) irrespective to (x_{ij}) , and the latter is determined only at each *terminal subproblems*. To construct such a branch-and-bound algorithm, we introduce a *subproblem* of BCMKP as follows. Let F_0 and F_1 be two subsets of M such that

$$K_0 \subseteq F_0, K_1 \subseteq F_1 \text{ and } F_0 \cap F_1 = \emptyset.$$

These represent the sets of knapsacks which are fixed at 0 and 1, respectively. We consider the following.

$P(F_0, F_1)$: maximize (1) subject to (2) - (5), and

$$y_i = 0, \forall i \in F_0, \quad y_i = 1, \forall i \in F_1. \quad (17)$$

Using the optimal λ^\dagger and μ^\dagger obtained in Section 2, the Lagrangian relaxation to this problem is

1 $\bar{P}(F_0, F_1)$: maximize (15) subject to $x_j, y_i \in \{0, 1\}, \forall j \in N, \forall i \in M$ and (17).
2

3 Let $z^*(F_0, F_1)$ and $\bar{z}(F_0, F_1)$ be the optimal objective values to these problems, respec-
4 tively. Clearly, $\bar{z}(F_0, F_1)$ gives an upper bound to $z^*(F_0, F_1)$, i.e., $z^*(F_0, F_1) \leq \bar{z}(F_0, F_1)$; and
5 we have
6

$$7 \quad \bar{z}(F_0, F_1) = \sum_{j=1}^n \theta_j^+ + \sum_{i \in F_1} \eta_i + \sum_{i \in U} \eta_i^+ + \mu^\dagger B, \quad (18)$$

8 where $U := M \setminus (F_0 \cup F_1)$ is the set of unfixed knapsacks. Then, if we have an *incumbent*
9 lower bound \underline{z} satisfying $\bar{z}(F_0, F_1) \leq \underline{z}$, we can *terminate* subproblem $P(F_0, F_1)$.
10

11 Other conditions for pruning subproblems are *feasibility* and *dominance*. First of all, if
12 the total cost of accepted knapsacks is larger than the budget, i.e., if $\sum_{i \in F_1} f_i > B$, $P(F_0, F_1)$ is
13 infeasible, and thus terminated. Next, if we have knapsacks $(i_0, i_1) \in F_0 \times F_1$ such that $c_{i_0} \geq c_{i_1}$
14 and $f_{i_0} \leq f_{i_1}$, $P(F_0, F_1)$ is again terminated. Indeed, if such a pair (i_0, i_1) exists, we can define
15 a subproblem $P(F'_0, F'_1)$ by exchanging the role of these knapsacks as $F'_0 := F_0 \cup \{i_1\} \setminus \{i_0\}$
16 and $F'_1 := F_1 \cup \{i_0\} \setminus \{i_1\}$. Then, $P(F_0, F_1)$ is dominated by $P(F'_0, F'_1)$, since all the feasible
17 solutions of $P(F_0, F_1)$ are feasible to $P(F'_0, F'_1)$.
18

19 If $P(F_0, F_1)$ is not terminated by any of these criteria, and in addition if U is non-empty,
20 we pick up a knapsack $i \in U$ and generate two subproblems of $P(F_0, F_1)$ as $P(F_0 \cup \{i\}, F_1)$
21 and $P(F_0, F_1 \cup \{i\})$. On the other hand, if $U = \emptyset$, $P(F_0, F_1)$ is a *terminal* subproblem. Here,
22 $P(F_0, F_1)$ is an MKP with respect to the set of knapsacks F_1 . An upper bound to this sub-
23 problem can be obtained by solving the following 0-1 knapsack problem, which is obtained
24 by replacing the set of knapsacks with a single knapsack of capacity $\sum_{i \in F_1} c_i$.
25

$$26 \quad \begin{aligned} & \text{maximize} && \sum_{j \in N} p_j x_j \\ & \text{subject to} && \sum_{j \in N} w_j x_j \leq \sum_{i \in F_1} c_i, \\ & && x_j \in \{0, 1\}, j \in N. \end{aligned}$$

27 Linear relaxation gives an upper bound, called the *Dantzig bound* [1], to this problem as
28

$$29 \quad \bar{z}_{\text{term}}(F_0, F_1) = \lfloor P_{b-1} + \frac{c_b}{w_b} (\sum_{i \in F_1} c_i - W_{b-1}) \rfloor, \quad (19)$$

30 and if $\bar{z}_{\text{term}}(F_0, F_1) \leq \underline{z}$, $P(F_0, F_1)$ is also terminated. Otherwise, we solve this MKP by calling
31 MULKNAP [13], and obtain the optimal $z^*(F_0, F_1)$. If this is better than the incumbent lower
32 bound \underline{z} , we update this as $\underline{z} \leftarrow z^*(F_0, F_1)$.
33

34 Then, we can construct a branch-and-bound algorithm to solve $P(F_0, F_1)$ as a recursive
35 procedure. The algorithm starts with the initial lower bound \underline{z} (obtained in section 2.2) and
36 $(F_0, F_1) := (K_0, K_1)$, and in termination produces an optimal solution to BCMKP. However, in
37 implementing the branch-and-bound algorithm, we have to specify the *strategy* for the choice
38 of the branching knapsack i , as well as the method to traverse the branch-and-bound tree. As
39 for the branching knapsack, under assumption \mathbf{A}_3 , we pick up the unfixed knapsack of the
40 smallest index, i.e., $i := \min\{k | k \in U\}$, and call $P(F_0, F_1 \cup \{i\})$ recursively before calling
41 $P(F_0 \cup \{i\}, F_1)$. This means that we examine subproblems with more knapsacks fixed at 1
42 earlier than others. For the latter we employ the *depth-first search* [14], since other methods
43 (such as the *breadth-first* or *best-first* methods) are usually heavy in memory requirements.
44 The algorithm is as follows.
45

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35

Algorithm SOLVE-BCMKP(F_0, F_1)

Input: $F_0, F_1 \subseteq M$, $F_0 \cap F_1 = \emptyset$, and an incumbent lower bound \underline{z} .

Step 1 (Check feasibility and dominance).

If $\sum_{i \in F_1} f_i > B$, or there exists a pair $(i_0, i_1) \in F_0 \times F_1$ such that $c_{i_0} \geq c_{i_1}$ and $f_{i_0} \leq f_{i_1}$, return.

Step 2 (Generalized pegging test).

If $\bar{z}(F_0, F_1) \leq \underline{z}$, return.

Step 3 (Terminal node).

If $U := M \setminus (F_0 \cup F_1) \neq \emptyset$, go to step 4. Otherwise do:

1. (Dantzig bound) If $\bar{z}_{\text{term}}(F_0, F_1) \leq \underline{z}$, return.
2. ($P(F_1, F_1)$ is an MKP).
 - Solve this problem using MULKNAP and obtain the optimal objective value $z^*(F_0, F_1)$.
 - If $z^*(F_0, F_1) > \underline{z}$, update the incumbent by $\underline{z} \leftarrow z^*(F_0, F_1)$.
 - return.

Step 4 (Branching).

1. Pick up the smallest $i \in U$.
2. Call SOLVE-BCMKP($F_0, F_1 \cup \{i\}$).
3. Call SOLVE-BCMKP($F_0 \cup \{i\}, F_1$).
4. return.

36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

Example 4. Applying SOLVE-BCMKP to the instance of Example 1, 28 subproblems (nodes) are generated, of which 7 are terminated due to infeasibility, and 6 due to upper bounds. 2 MKPs are solved by MULKNAP, and we obtain the optimal $z^* = 3916$. The same objective value is obtained by solving this BCMKP directly using CPLEX 11.1. ■

5 Numerical experiments

We evaluate the performance of the branch-and-bound algorithm of the previous section through a series of numerical experiments. We implement the algorithm in ANSI C language and conduct computation on an Dell Precision T7400 workstation (CPU: Xeon X5482 Quad-Core \times 2, 3.20GHz).

5.1 Design of experiments

Instances are prepared within the range of $200 \leq n \leq 160000$ and $5 \leq m \leq 150$ according to the following scheme. The weight w_j is distributed uniformly random over the integer interval $[10, 1000]$, and profit p_j is related to w_j in the following way (See Figure 2).

- Uncorrelated case (UNCOR): uniformly random over $[10, 1000]$, independent of w_j .

- Weakly correlated case (WEAK): uniformly random over $[w_j, w_j + 200]$.
- Strongly correlated case (STRONG): $p_j := w_j + 20$

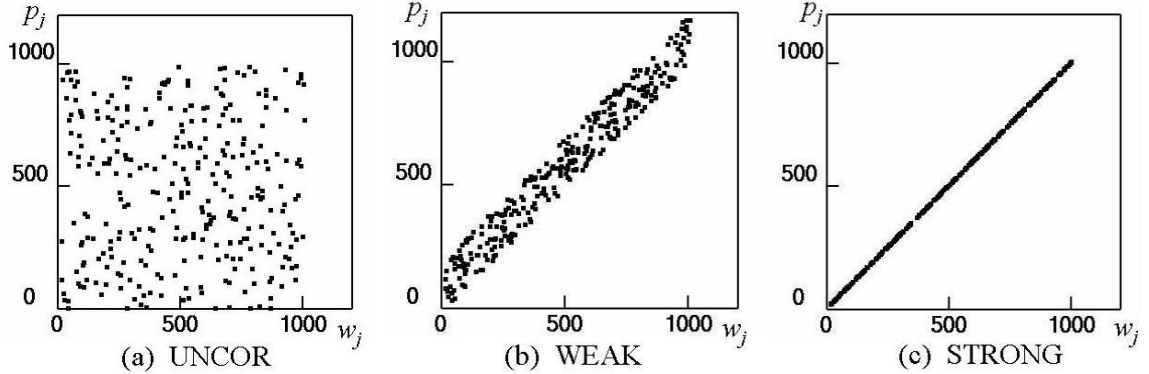


Figure 2: Correlation of w_j and p_j .

Knapsack capacity c_i is determined by $c_i = \lfloor 500n \cdot \alpha \cdot \xi_i \rfloor$, where (ξ_i) is uniformly distributed over the simplex $\{(\xi_1, \dots, \xi_m) \mid \sum_{i=1}^m \xi_i = 1, \xi_i \geq 0\}$, and α is a parameter to control the ratio of items that can be accepted into the knapsacks. Since average weight of items is approximately 500, $\alpha = 0.50$ means that about a half of all the items can be accommodated in the knapsacks. Knapsack cost is given by $f_i := \rho_i c_i$, where ρ_i is uniformly random over $[0.5, 1.5]$, and the budget B is chosen as $B = \beta \sum_{i=1}^m f_i$. Here β is another parameter that controls the ratio of the budget B over the total cost of knapsacks.

5.2 Comparison against MIP solvers

Table 5 summarizes the computation of small instances with parameters $\alpha = 0.5$ and $\beta = 0.6$, $n = 10 \sim 40$ and $m = 3 \sim 12$ using MIP solver CPLEX 11.1 [7] and branch-and-bound method of section 4. For each correlation type and values of n and m , 10 random instances are generated and solved. Here shown are the number of instances solved to optimality within a fixed CPU time (#solved), and the average CPU time in seconds. We set the time-limit of computation at 1800 CPU seconds, and if computation is truncated due to this time-limit, the CPU time for this instance is interpreted as 1800 seconds in computing averages.

From Table 5 we see that commercial solvers are able to solve only very small instances within the time-limit, while branch-and-bound solved all these problems within 30 seconds.

5.3 Large instances

Table 6 gives the results of computation of upper and lower bounds as well as the pegging test for larger instances with $n = 20000 \sim 160000$ and $m = 50 \sim 150$. The table shows the upper and lower bounds (\bar{z} and \underline{z} , respectively), and the column of 'Error(%)' gives their relative errors defined by $100 \cdot (\bar{z} - \underline{z}) / \underline{z}$. Applying the pegging test, some knapsacks are fixed either at 0 or 1, and m' shows the number of unfixed knapsacks, i.e., $m' := |M \setminus (K_0 \cup K_1)|$. 'CPU₁' is the CPU time in seconds to compute upper and lower bounds, as well as to carry out

Table 5: Comparison against MIP solvers.

Correlation	n	m	branch-and-bound		CPLEX 11.1	
			#solved	CPU _{sec}	#solved	CPU _{sec}
UNCOR	10	3	10	0.00	10	0.01
	20	6	10	0.00	10	0.08
	30	9	10	1.66	10	8.81
	40	12	10	14.31	6	845.00
WEAK	10	3	10	0.00	10	0.01
	20	6	10	0.00	10	0.48
	30	9	10	0.34	10	107.03
	40	12	10	16.84	0	1800.00
STRONG	10	3	10	0.00	10	0.01
	20	6	10	0.00	10	2.54
	30	9	10	0.04	9	374.06
	40	12	10	0.66	2	1637.38

the pegging test. From this table, we observe that the pegging works effectively in reducing the number of knapsacks. However, we found that it is less effective for reducing the size of n .

Table 7 is the results of the branch-and-bound algorithm for various instances. This table shows the optimal value (z^*), the number of the generated nodes in the branch-and-bound algorithm (#nodes), the number of pruned subproblems due to infeasibility (#infeas), or due to upper bounds (#ub), the number of terminal MKPs solved by calling MULKNAP (#Pis), and the CPU time in seconds (CPU₂). Each row is again the average over 10 random instances.

Except for some cases, the branch-and-bound method solved BCMKPs exactly by invoking MULKNAP only a few times and within a few minutes. We conclude that the branch-and-bound algorithm is very successful for these large instances.

Table 6: Bounding and pegging results for large instances.

Correlation	m	$n(\times 10^4)$	$\bar{z}(\times 10^7)$	$\underline{z}(\times 10^7)$	Error(%)	m'	CPU ₁
UNCOR	50	2	0.67839	0.67776	0.09	13.6	0.01
		4	1.36165	1.36010	0.11	16.2	0.02
		8	2.71760	2.71492	0.10	14.9	0.05
		16	5.43095	5.42441	0.12	16.0	0.10
	100	2	0.68241	0.68223	0.03	19.3	0.01
		4	1.36177	1.36125	0.04	23.6	0.02
		8	2.71905	2.71781	0.05	26.7	0.05
		16	5.42959	5.42713	0.05	23.4	0.10
	150	2	0.68306	0.68299	0.01	17.8	0.01
		4	1.36191	1.36173	0.01	21.1	0.02
		8	2.71811	2.71749	0.02	31.9	0.05
		16	5.44194	5.44055	0.02	31.2	0.10
WEAK	50	2	0.48252	0.48176	0.16	13.6	0.01
		4	0.97220	0.97032	0.20	16.2	0.02
		8	1.93985	1.93660	0.17	14.9	0.05
		16	3.87370	3.86576	0.21	16.0	0.11
	100	2	0.48740	0.48718	0.04	19.3	0.01
		4	0.97232	0.97168	0.07	23.6	0.02
		8	1.94149	1.93998	0.08	26.7	0.05
		16	3.87202	3.86903	0.08	23.4	0.11
	150	2	0.48819	0.48810	0.02	17.8	0.01
		4	0.97251	0.97229	0.02	21.1	0.03
		8	1.94033	1.93959	0.04	31.9	0.06
		16	3.88695	3.88525	0.04	31.3	0.11
STRONG	50	2	0.37426	0.37326	0.27	17.1	0.48
		4	0.74238	0.74094	0.19	14.2	1.08
		8	1.49757	1.49432	0.22	16.2	2.09
		16	2.98705	2.98142	0.19	14.9	6.76
	100	2	0.37287	0.37264	0.06	20.7	0.61
		4	0.75081	0.75034	0.06	23.0	1.57
		8	1.49777	1.49652	0.09	26.1	2.89
		16	2.98985	2.98719	0.09	27.2	8.63
	150	2	0.37527	0.37520	0.02	16.2	0.59
		4	0.75219	0.75203	0.02	18.5	0.84
		8	1.49811	1.49746	0.04	31.2	2.49
		16	2.98785	2.98645	0.05	34.8	3.45

Table 7: Branch-and-bound results for large instances.

Correlation	m	$n(\times 10^4)$	$z^*(\times 10^7)$	#nodes	#infeas	#ub	#Pis	CPU ₂
UNCOR	50	2	0.67793	1525.8	178.1	583.7	2.1	0.42
		4	1.36067	2986.0	476.9	1014.6	2.5	1.57
		8	2.71614	769.0	140.5	243.1	1.9	0.82
		16	5.42675	1086.0	160.2	380.9	2.9	2.36
	100	2	0.68226	7414.8	763.9	2942.8	1.7	2.01
		4	1.36153	7081.0	970.5	2567.8	3.2	3.75
		8	2.71849	8054.8	1059.9	2965.1	3.4	8.58
		16	5.42826	4840.4	939.5	1479.1	2.6	9.48
	150	2	0.68300	3254.4	490.5	1136.1	1.6	0.74
		4	1.36178	4353.0	431.4	1744.1	2.0	2.46
		8	2.71782	9085.8	858.2	3683.4	2.3	10.12
		16	5.44151	11524.8	1443.7	4315.9	3.8	24.38
WEAK	50	2	0.48197	1525.2	177.9	583.6	2.1	0.40
		4	0.97101	2988.8	477.9	1015.0	2.5	1.56
		8	1.93808	769.0	140.5	243.1	1.9	0.82
		16	3.86860	1088.8	160.5	382.0	2.9	2.37
	100	2	0.48722	7416.2	763.8	2943.6	1.7	1.94
		4	0.97202	7085.8	972.3	2568.4	3.2	3.71
		8	1.94080	8049.0	1061.3	2960.8	3.4	8.49
		16	3.87040	4841.4	939.2	1479.9	2.6	9.45
	150	2	0.48812	3260.2	491.5	1138.0	1.6	0.81
		4	0.97235	4370.4	431.7	1752.5	2.0	2.44
		8	1.94000	9121.8	863.4	3696.2	2.3	10.05
		16	3.88642	11559.4	1444.2	4332.7	3.8	24.37
STRONG	50	2	0.37365	1894.6	356.1	589.8	2.4	1.53
		4	0.74142	1586.0	193.9	597.7	2.4	2.75
		8	1.49551	2946.8	475.4	996.5	2.5	8.91
		16	2.98398	767.4	140.5	242.3	1.9	13.17
	100	2	0.37271	3317.6	436.3	1221.0	2.5	1.83
		4	0.75050	10080.6	1173.8	3863.7	3.8	8.81
		8	1.49727	7365.6	1027.8	2652.3	3.7	17.10
		16	2.98867	8526.4	1093.5	3167.2	3.5	42.15
	150	2	0.37520	3685.8	325.7	1517.1	1.1	1.53
		4	0.75206	4666.6	613.7	1718.6	2.0	4.30
		8	1.49783	7787.2	965.1	2925.4	4.1	21.62
		16	2.98725	9468.6	901.8	3830.8	2.7	31.81

5.4 Sensitivity analysis

We now examine sensitivity of the algorithm with respect to the parameters α and β , which have been fixed at $\alpha = 0.5$ and $\beta = 0.6$ in the previous experiments. Tables 8 and 9 give the results of sensitivity analysis, where we show the result of WEAK case with $n = 80000$ and $m = 100$.

Table 8: Sensitivity analysis on α (WEAK, $n = 80000, m = 100, \beta = 0.6$).

α	0.3	0.4	0.5	0.6	0.7	0.8
Error(%)	0.07	0.08	0.08	0.08	0.08	0.08
$z^*(\times 10^7)$	1.2569	1.6027	1.9408	2.2728	2.5989	2.9193
#nodes	8044.8	7996.6	8049.0	8236.4	8341.0	8467.0
CPU ₂	0.98	0.66	0.67	0.70	0.70	0.72

Table 9: Sensitivity analysis on β (WEAK, $n = 80000, m = 100, \alpha = 0.5$).

β	0.3	0.4	0.5	0.6	0.7	0.8
Error(%)	0.13	0.10	0.08	0.08	0.11	0.06
$z^*(\times 10^7)$	1.2505	1.5050	1.7323	1.9408	2.1340	2.3139
#nodes	3066.8	4124.4	7410.2	8049.0	3968.8	3222.0
CPU ₂	0.26	0.35	0.62	0.67	0.32	0.29

From these tables, we see that the optimal objective value (z^*) increases almost linearly with the increase of α as well as of β . However, these parameters do not have much effect either on the relative errors between the upper and lower bounds, the number of the subproblems generated in the branch-and-bound process, or its CPU time.

5.5 Weakness of the algorithm

It is known that MULKNAP is less effective in solving MKPs with relatively few items per knapsack. Here we show the case of BCMKP with fewer number of items, i.e., $200 \leq n \leq 500$. Tables 10 and 11 summarize the results for these cases. From Table 10, we see that the relative errors are higher than in Table 6, especially for smaller n and larger m such as $n = 200$ and $m = 150$. Correspondingly, the pegging test is less effective for these cases.

Table 11 shows the result of the branch-and-bound. We truncated computation at the time-limit of 1800 seconds, and in such a case we took as z^* the best incumbent value at that time, and the CPU time was set to 1800. Again, we were unable to solve instance with smaller n and larger m (such as $n = 200$ and $m = 150$) within the time-limit. We conclude that branch-and-bound inherits the weakness of MULKNAP, and have difficulty for problems with relatively small n per knapsack.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

Table 10: Bounding and pegging results for small instance.

Correlation	m	n	$\bar{z}(\times 10^5)$	$\underline{z}(\times 10^5)$	Error(%)	m'	
UNCOR	50	200	0.68195	0.68073	0.18	20.7	
		300	1.01466	1.01355	0.11	15.5	
		400	1.35890	1.35772	0.09	14.6	
		500	1.71594	1.71459	0.08	12.6	
	100	200	0.67888	0.66438	2.20	100.0	
		300	1.01771	1.01354	0.41	80.6	
		400	1.35767	1.35714	0.04	24.8	
		500	1.71465	1.71407	0.03	22.8	
	150	200	0.67994	0.61724	10.29	150.0	
		300	1.01576	0.99519	2.07	150.0	
		400	1.35929	1.34522	1.06	145.8	
		500	1.71317	1.71250	0.04	41.2	
	WEAK	50	200	0.48864	0.48753	0.23	16.8
			300	0.72684	0.72570	0.16	13.9
			400	0.97050	0.96923	0.13	13.1
			500	1.21957	1.21810	0.12	11.5
100		200	0.48493	0.47571	1.94	99.9	
		300	0.73072	0.72892	0.25	55.4	
		400	0.96900	0.96860	0.04	18.5	
		500	1.21767	1.21711	0.05	19.6	
150		200	0.48631	0.44993	8.15	150.0	
		300	0.72825	0.71694	1.58	150.0	
		400	0.97096	0.96667	0.45	125.5	
		500	1.21589	1.21547	0.04	29.5	
STRONG		50	200	0.37262	0.37184	0.21	15.2
			300	0.56287	0.56205	0.15	10.6
			400	0.75498	0.75373	0.17	11.7
			500	0.93543	0.93366	0.19	13.6
	100	200	0.37450	0.37121	0.89	91.0	
		300	0.56207	0.56150	0.10	27.5	
		400	0.74866	0.74827	0.05	16.5	
		500	0.93999	0.93946	0.06	18.7	
	150	200	0.37592	0.35959	4.59	150.0	
		300	0.56384	0.56016	0.66	139.7	
		400	0.75124	0.75070	0.07	44.6	
		500	0.94049	0.94002	0.05	33.5	

Table 11: Branch-and-bound result for small instances.

Correlation	m	n	$z^*(\times 10^5)$	#nodes	#infeas	#ub	#Pis	CPU ₂	
UNCOR	50	200	0.68102	3244.1	629.9	987.0	5.0	182.83	
		300	1.01355	2701.0	359.8	988.8	2.9	0.00	
		400	1.35772	1406.4	157.2	545.0	2.0	0.00	
		500	1.71459	1307.8	122.8	530.3	1.8	0.00	
	100	200	0.66438	128.4	28.4	0.0	0.0	1800.00	
		300	1.01362	113.4	29.6	0.0	0.1	1800.00	
		400	1.35715	161249.6	22250.1	58298.1	77.6	0.25	
		500	1.71407	12515.6	1053.8	5199.4	5.6	0.01	
	150	200	0.61724	192.9	42.9	0.0	0.0	1800.00	
		300	0.99519	191.2	41.2	0.0	0.0	1800.00	
		400	1.34522	188.6	42.8	0.0	0.0	1800.00	
		500	1.71250	64457.2	5625.4	26582.2	13.0	900.09	
	WEAK	50	200	0.48770	1388.5	328.1	362.9	2.1	546.35
			300	0.72570	1620.6	216.6	593.2	1.5	0.00
			400	0.96923	1027.4	120.1	393.0	1.6	0.00
			500	1.21810	991.6	89.1	406.4	1.3	0.00
100		200	0.47571	128.3	28.4	0.0	0.0	1800.00	
		300	0.72892	78.1	22.7	0.0	0.0	1800.00	
		400	0.96860	9549.6	832.1	3940.0	3.7	0.01	
		500	1.21711	4639.6	484.7	1834.6	1.5	0.00	
150		200	0.44993	192.9	42.9	0.0	0.0	1800.00	
		300	0.71694	191.2	41.2	0.0	0.0	1800.00	
		400	0.96667	167.4	41.9	0.0	0.0	1800.00	
		500	1.21554	12181.6	806.6	5276.5	2.2	1080.05	
STRONG		50	200	0.37192	1197.0	138.4	458.8	2.3	0.01
			300	0.56205	794.6	122.8	274.4	1.1	0.00
			400	0.75373	815.0	132.3	274.9	1.3	0.00
			500	0.93366	1189.8	174.5	420.3	1.1	0.01
	100	200	0.37121	119.6	28.6	0.0	0.0	1800.00	
		300	0.56164	16286.6	2273.6	5861.1	9.6	30.82	
		400	0.74830	4432.8	595.5	1618.2	3.7	0.17	
		500	0.93952	8602.6	751.7	3545.2	5.4	0.08	
	150	200	0.35959	190.8	40.8	0.0	0.0	1800.00	
		300	0.56016	181.4	41.7	0.0	0.0	1800.00	
		400	0.75079	55412.2	7969.2	19682.0	48.2	735.37	
		500	0.94021	22567.0	2542.7	8735.5	6.3	0.83	

6 Conclusion

We have formulated the budget-constrained multiple knapsack problem, and presented an algorithm to solve this problem to optimality. By combining the Lagrangian relaxation and pegging test with the branch-and-bound method that solves MKP at each terminal nodes by calling MULKNAP, we were able to solve almost all BCMKPs with up to $n = 160000$ items and $m = 150$ knapsacks within a few minutes in an ordinary computing environment. However instances with smaller n and larger m remain hard to be solved exactly.

Acknowledgement.

MULKNAP was downloaded from Prof. D. Pisinger's homepage (<http://www.diku.dk/hjemmesider/ansatte/pisinger/>) for which the authors express sincere gratitude.

References

- [1] G.B. Dantzig, Discrete variable extremum problems, *Operations Research* 5 (1957) 266-277.
- [2] R.S. Dembo, P.L. Hammer, A reduction algorithm for knapsack problems, *Methods of Operations Research* 36 (1980) 49-60.
- [3] M. Fisher, The Lagrangian relaxation method for solving integer programming problems, *Management Science* 27 (1981) 1-18.
- [4] M.R. Garey, D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman and Company, San Francisco, 1979.
- [5] E. Horowitz, S. Sahni, Computing partitions with application to the knapsack problem, *Journal of the ACM* 21 (1974) 277-292.
- [6] M.S. Hung, J.C. Fisk, An algorithm for 0-1 multiple knapsack problems, *Naval Research Logistics Quarterly* 25 (1978) 571-579.
- [7] ILOG, CPLEX 11.1, <http://www.ilog.com/products/cplex>, 2009.
- [8] G.P. Ingargiola, F.F. Korsh, Reduction algorithms for zero-one single knapsack problems, *Management Science* 20 (1973) 460-463.
- [9] H. Kellerer, U. Pferschy, D. Pisinger, *Knapsack Problems*, Springer, Berlin, 2004.
- [10] S. Martello, P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*, John Wiley & Sons, Chichester, 1990.
- [11] R.M. Nauss, *Parametric Integer Programming*, University of Missouri Press, Columbia, MI, 1979.
- [12] D. Pisinger, An expanding-core algorithm for the exact 0-1 knapsack problem, *European Journal of Operational Research* 87 (1995) 175-187.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

[13] D. Pisinger, An exact algorithm for large multiple knapsack problems, *European Journal of Operational Research* 114 (1999) 528-541.

[14] R. Sedgewick, *Algorithm in C*, Third Ed., Addison-Wosley, Reading, 1998.

[15] L. Wolsey, *Integer Programming*, John Wiley & Sons, New York, 1998.

[16] T. Yamada, T. Takeoka, An exact algorithm for the fixed-charge multiple knapsack problem, *European Journal of Operational Research* 192 (2009) 700-705.