An Exact Algorithm for Large Multiple Knapsack Problems

David Pisinger

Dept. of Computer Science, University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen, Denmark

pisinger@diku.dk

Abstract

The Multiple Knapsack Problem is the problem of assigning a subset of n items to m distinct knapsacks, such that the total profit sum of the selected items is maximized, without exceeding the capacity of each of the knapsacks. The problem has several applications in naval as well as financial management.

A new exact algorithm for the Multiple Knapsack Problem is presented, which is specially designed for solving large problem instances. The recursive branch-and-bound algorithm applies surrogate relaxation for deriving upper bounds, while lower bounds are obtained by splitting the surrogate solution into the m knapsacks by solving a series of Subset-sum Problems. A new separable dynamic programming algorithm is presented for the solution of Subset-sum Problems, and we also use this algorithm for tightening the capacity constraints in order to obtain better upper bounds.

The developed algorithm is compared to the MTM algorithm by Martello and Toth, showing the benefits of the new approach. A surprising result is, that large instances with $n = 100\,000$ items may be solved in less than a second, and the algorithm has a stable performance even for instances with coefficients in a moderately large range.

Keywords: Programming: Integer, Knapsack Problem, Loading, Programming: Dynamic.

1 Introduction

We consider the problem where n given items should be packed in m knapsacks of distinct *capacities* c_i, $i = 1, \ldots, m$. Each *item* j has an associated *profit* p_j and *weight* w_j, and the problem is to select m disjoined subsets of items, such that subset i fits into capacity c_i and the total profit of the selected items is maximized. Thus we may formally define the *0-1 Multiple Knapsack Problem* (*MKP*) as the ILP problem

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij} \\ \text{subject to} \quad & \sum_{j=1}^{n} w_j x_{ij} \leq c_i, \quad i = 1, \ldots, m, \\ & \sum_{i=1}^{m} x_{ij} \leq 1, \qquad j = 1, \ldots, n, \\ & x_{ij} \in \{0,1\}, \qquad i = 1, \ldots, m, \; j = 1, \ldots, n, \end{aligned} \tag{1}$$

where $x_{ij} = 1$ if item j is assigned to knapsack i, and $x_{ij} = 0$ otherwise. It is usual to assume that the coefficients p_j, w_j and c_i are positive integers, and to avoid trivial cases we demand that

$$\max_{j=1,\ldots,n} w_j \leq \max_{i=1,\ldots,m} c_i \qquad \min_{i=1,\ldots,m} c_i \geq \min_{j=1,\ldots,n} w_j \qquad \sum_{j=1}^{n} w_j \geq \max_{i=1,\ldots,m} c_i. \tag{2}$$

The first assumption ensures that each item j fits into at least one knapsack as otherwise it may be removed from the problem. If the second inequality is violated, then we may discount the smallest knapsack, as no items fit into it. Finally the last inequality avoids a trivial solution where all items fit into the largest of the knapsacks.

There are several applications for MKP, as the problem directly reflects a situation of loading m ships or vehicles. Martello and Toth [10] proposed to use MKP for deciding how to load m liquids into n tanks, when the liquids may not be mixed, and MKP is the immediate generalization of the well-known Cutting-stock Problem to cases where stocks may have different lengths. This problem has several applications in the paper and steel industry [16].

The Multiple Knapsack Problem is $\mathcal{NP}$-hard in the strong sense, and thus any dynamic programming approach would result in strictly exponential time bounds. Several branch-and-bound algorithms for MKP have thus been presented during the last two decades, among which we should mention Hung and Fisk [7], Martello and Toth [8], Neebe and Dannenbring [11], and Christofides, Mingozzi and Toth [3]. The first two are best suited for problems where several items are filled into each knapsack, while the last two are designed for problems with many knapsacks and few items.

This paper follows the first direction of research, as it is devoted to large problem instances where the quotient n/m is relatively large. The presented algorithm is based on the MTM framework by Martello and Toth [9] which has been improved in several respects: Lower bounds are derived by solving a series of Subset-sum Problems, and Subset-sum Problems are also used for tightening the capacity constraints of each knapsack. We use a well-performing 0-1 Knapsack Algorithm for deriving upper bounds through surrogate relaxation, and a new separable dynamic programming algorithm is presented for solving the individual Subset-sum Problems. Moreover efficient reduction rules are applied to determine which items cannot be packed into any of the knapsacks.

In Section 2 we will present different upper bounds for the Multiple Knapsack Problem, and use these observations for a general discussion of bound-and-bound algorithms in

Section 3. The discussion leads to an improved algorithm described in Section 4. Section 5 shows how lower bounds may be achieved by solving a series of Subset-sum Problems, and how the capacity constraints in MKP may be tightened in order to obtain better upper bounds. Section 6 considers some reduction algorithms for the problem, and finally Section 7 compares the presented MULKNAP algorithm with the MTM algorithm by Martello and Toth [9]. A first version of this paper was presented in [14].

2 Upper bounds

Upper bounds for the Multiple Knapsack Problem may be derived by relaxing some of the side-constraints, where Surrogate, Lagrangian and linear relaxations are the most common relaxations.

By relaxing the constraint $\sum_{i=1}^{m} x_{ij} \leq 1,\ j = 1, \ldots, n$ in formulation (1), Martello and Toth [9] showed that the *Lagrangian relaxed* problem may be decomposed into m independent 0-1 Knapsack Problems. There is however no optimal choice of the multipliers $\{\lambda_j\}$ for the relaxation, thus usually a subgradient optimization technique must be used, making these bounds computationally expensive to derive.

For the surrogate relaxation optimal multipliers may however be found analytically: Let $\pi_1, \ldots, \pi_m$ be some positive multipliers, then the *Surrogate relaxed Multiple Knapsack Problem* (*SMKP*) becomes:

$$\begin{array}{lll} \text{maximize} & \displaystyle\sum_{i=1}^{m}\sum_{j=1}^{n} p_j x_{ij} & \\ \text{subject to} & \displaystyle\sum_{i=1}^{m} \pi_i \sum_{j=1}^{n} w_j x_{ij} \leq \sum_{i=1}^{m} \pi_i c_i, & \\ & \displaystyle\sum_{i=1}^{m} x_{ij} \leq 1, & j = 1, \ldots, n, \\ & x_{ij} \in \{0, 1\}, & i = 1, \ldots, m,\ j = 1, \ldots, n. \end{array} \tag{3}$$

The best choice of multipliers π_i are those, that produce the smallest objective value in (3). Martello and Toth [9] proved that for any instance of MKP, the optimal choice of multipliers $\pi_1, \ldots, \pi_m$ is $\pi_i = k$ for $i = 1, \ldots, m$, where k is a positive constant. With this choice of multipliers SMKP becomes the following ordinary 0-1 Knapsack Problem

$$\begin{array}{lll} \text{maximize} & \displaystyle\sum_{j=1}^{n} p_j x'_j & \\ \text{subject to} & \displaystyle\sum_{j=1}^{n} w_j x'_j \leq c, & \\ & x'_j \in \{0, 1\}, & j = 1, \ldots, n, \end{array} \tag{4}$$

where the introduced decision variables $x'_j = \sum_{i=1}^{m} x_{ij}$ indicate whether item j is chosen for any of the knapsacks $i = 1, \ldots, m$, and $c = \sum_{i=1}^{m} c_i$ may be seen as the capacity of the united knapsacks.

Linear relaxation is a third common technique for deriving upper bounds. We relax the constraint $x_{ij} \in \{0,1\}$ in (1) to $0 \le x_{ij} \le 1$, for $i = 1,\ldots,m$, $j = 1,\ldots,n$. Martello and Toth [10] proved that the objective value of an optimal solution to the linear relaxed MKP is the same as that of an optimal solution to the linear relaxed SMKP. Thus to find the objective value of the linear MKP, we may simply use the Dantzig upper bound of the corresponding 0-1 Knapsack Problems: Assume that the items are ordered according to nonincreasing profit-to-weight ratios $p_1/w_1 \ge p_2/w_2 \ge \ldots \ge p_n/w_n$ and let the *break item* b be given by $b = \min\left\{j : \sum_{i=1}^{j} w_i \ge c\right\}$, where $c = \sum_{i=1}^{m} c_i$. Then the Dantzig upper bound [4] becomes

$$u_{\mathrm{MKP}} = \sum_{j=1}^{b-1} p_j + \left\lfloor \left(c - \sum_{j=1}^{b-1} w_j\right) p_b/w_b \right\rfloor. \tag{5}$$

The linear upper bound may be derived in $O(n)$ time using the technique of Balas and Zemel [1], thus being considerably faster to determine than the previous two bounds. Computational experiments have however shown [10] that the linear bound generally is too weak to efficiently cut off branches in an enumerative algorithm.

3 Bound-and-Bound algorithm

Martello and Toth [9] focused on three aspects that make it difficult to solve MKP:

- Generally it is difficult to verify feasibility of an upper bound obtained either by surrogate relaxation or Lagrangian relaxation.
- The branch-and-bound algorithm needs good lower bounds for fathoming nodes in the enumeration.
- We need some knowledge to guide the branching towards good feasible solutions.

In order to evade these problem, Martello and Toth proposed a *bound-and-bound* algorithm for MKP where at each node of the branching tree, not only an upper bound, but also a lower bound is derived. This technique is well suited for problems where it is easy to find a fast heuristic solution which yields good lower bounds, and where it is difficult to verify feasibility of the upper bound.

The MTM algorithm by Martello and Toth derives upper bounds by solving the surrogate relaxed problem (3) while the lower bounds are found by solving m individual 0-1 Knapsack Problems as follows: First knapsack $i = 1$ is filled optimally, the chosen variables are removed from the problem, and the next knapsack $i = 2$ may be filled. The process is repeated until all m knapsacks have been filled. The branching scheme follows this greedy solution, as Martello and Toth argue that a greedy solution is better to guide the branching process than individual choices at each branching node. Thus at each node we fork in two branching nodes, the one assigning the next item j of a greedy solution to the chosen knapsack i, while the other branch excludes item j from knapsack i.

4 New algorithm

The concept of bound-and-bound algorithms was intended for problems where feasibility of the upper bound solution is difficult to validate. But a solution to SMKP may actually be validated by solving a series of Subset-sum Problems, where we try to split the chosen items into the m knapsacks. If this attempt succeeds, our lower bound equals the upper bound, and we may immediately backtrack. Otherwise we have obtained a feasible solution which contains some (but not all) of the items selected by SMKP. This attempt seems more promising than the approach presented by Martello and Toth, since we consider the m individual packing problems as an integrated unit.

Furthermore we use Subset-sum Problems for tightening the capacity constraints corresponding to each knapsack: If we cannot fill the knapsack completely even if all items may be used, then the gap may be closed without excluding any optimal solution.

The branching scheme is based on a binary splitting where an item j is either assigned to knapsack i or excluded from the knapsack. The knapsacks are ordered such that $c_1 \leq c_2 \leq \ldots \leq c_m$ and we fill the smallest knapsack completely before starting to fill the next knapsack. At any stage, items $j \leq h$ have been fixed by the branching process, thus only items $j > h$ are considered when upper and lower bounds are determined. To keep track of which items are excluded from some knapsacks, we assign a variable d_j to each item j which says that the item may only be assigned to knapsacks $i \geq d_j$. The current solution vector is y, the lower bound z, while P and W are the profit and weight sums of the items that already have been fixed to a knapsack. Thus we have the following pseudocode sketch of the recursive branch-and-bound algorithm:

algorithm mulbranch$(h, P, W, c_1, \ldots, c_m)$;
Tighten the capacities c_i by solving m Subset-sum Problems defined on items $h+1, \ldots, n$.
Solve the surrogate relaxed problem (4) with capacity $c = \sum_{i=1}^{m} c_i$. Let x' be the solution
  to this problem with objective value u.
if $(P + u > z)$ **then** { *Upper bound test* }
  Sort the items given by x' according to nonincreasing weights.
  Split the solution x' in the m knapsacks by solving a series of Subset-sum Problems
    defined on items with $x'_j = 1$. Let y_{ij} be the optimal filling of c_i with corresponding
    profit sum z_i.
  Improve the heuristic solution for knapsacks i which were not filled completely.
  if $(P + \sum_{i=1}^{m} z_i > z)$ **then** Copy y to x, set $z \leftarrow P + \sum_{i=1}^{m} z_i$ **end if**;
end if;
if $(P + u > z)$ **then** { *Check the bound again since z may have been improved* }
  Reduce the items by using some upper bound tests, and swap the reduced items to the
    first positions, increasing h.
  Let i be the smallest knapsack with $c_i > 0$. Solve an ordinary 0-1 Knapsack Problem
    with $c = c_i$ defined on the free variables. The solution vector is x'. Choose the
    branching item ℓ as the item with largest profit-to-weight ratio among items $x'_j = 1$.
  Swap ℓ to position $h + 1$ and set $j \leftarrow h + 1$.

Let $y_{ij} \leftarrow 1$; { *Assign item j to knapsack i* }
Recursively call mulbranch($h+1, P+p_j, W+w_j, c_1, \ldots, c_i - w_j, \ldots, c_m$);
Let $y_{ij} \leftarrow 0$; { *Exclude item j from knapsack i* }
Set $d' \leftarrow d_j$; $d_j \leftarrow i+1$;
Recursively call mulbranch($h, P, W, c_1, \ldots, c_m$);
Find j again, and set $d_j \leftarrow d'$.
end if;

The surrogate relaxed problem is solved using the MINKNAP algorithm presented in Pisinger [13] which is based on dynamic programming. The algorithm has been modified such that the currently best objective value from MKP is used as an initial lower bound, and such that if several optimal solutions have the same objective value, we choose the solution which has smallest weight sum. The Subset-sum Problems are solved through separable dynamic programming which will be described in Section 5. In the same section we show how Subset-sum Problems may be used for tightening the capacity constraints, and how lower bounds are improved by solving ordinary 0-1 Knapsack Problems. Finally the reduction algorithm is described in Section 6.

The algorithm does not demand any special ordering of the variables, as the MINKNAP algorithm makes the necessary ordering itself. This however implies, that items are permuted at each call to MINKNAP thus the last line of procedure MULBRANCH cannot assume that item j is at the same position as before.

The main algorithm MULKNAP only needs to order the capacities and initialize a few variables before the branch-and-bound algorithm is called:

algorithm mulknap(n, m, p, w, x, c);
Order the capacities $c_1 \leq c_2 \leq \ldots \leq c_m$.
Set $d_j \leftarrow 1$ for $j = 1, \ldots, n$ and $x_{ij} \leftarrow 0$; $y_{ij} \leftarrow 0$; for $i = 1, \ldots, m,\ j = 1, \ldots, n$.
Set $z \leftarrow 0$ and call mulbranch($0, 0, 0, c_1, \ldots, c_m$);

5 The Subset-sum Problem

As previously mentioned lower bounds are obtained by splitting the chosen items in SMKP into the m knapsacks. This is done by solving a series of Subset-sum Problems, where the knapsacks are considered for increasing values of c_i. Small knapsacks are generally more difficult to solve than large ones, since very few items fit inside.

In the first iteration we fill the smallest knapsack c_1 as much as possible, and remove those items from the problem. Then the second smallest knapsack is filled, and we continue this way until all knapsacks have been considered. If all items could be split into the m knapsacks in this way, then the upper and lower bound of the current decision node are equal and we may immediately backtrack. Let $N = \{j : x'_j = 1\}$ be the set of items in an optimal solution x' to the 0-1 Knapsack Problem (4). Assuming that $N = \{1, \ldots, n\}$ (not to be confused with the index set of MKP) the individual Subset-sum Problems may be defined as:

$$\begin{aligned} \text{maximize} \quad & z = \sum_{j=1}^{n} w_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} w_j x_j \leq c, \\ & x_j \in \{0,1\}, \quad j = 1, \ldots, n. \end{aligned} \tag{6}$$

Each problem may be solved in $O(nc)$ time through dynamic programming by using the recursion by Bellman [2], or by using the balanced dynamic programming algorithm of Pisinger [12] which has time bound (nr) where $r = \max_{j=1}^{n} w_j < c$.

However our situation is quite special as for large instances of (6) we generally have several optimal solutions with $\sum_{j=1}^{n} w_j x_j = c$ while small instances demand most enumeration. Thus a specialized algorithm was developed based on the separability property observed by Horowitz and Sahni [6]: Let $b = \min\left\{j : \sum_{i=1}^{j} w_i > c\right\}$, be the break item of (6) and let $f_t(\tilde{c})$, $(b-1 \leq t \leq n,\ 0 \leq \tilde{c} \leq c)$ be an optimal solution to the above problem restricted to variables $b, \ldots, t$ as follows:

$$f_t(\tilde{c}) = \max \left\{ \begin{array}{l} \sum_{j=b}^{t} w_j x_j : \\ \sum_{j=b}^{t} w_j x_j \leq \tilde{c}, \\ x_j \in \{0,1\},\ j = b, \ldots, t. \end{array} \right. \tag{7}$$

Let $g_s(\tilde{c})$, $(1 \leq s \leq b,\ 0 \leq \tilde{c} \leq c)$ be an optimal solution to the problem defined on variables $s, \ldots, b-1$ with the additional constraint $x_j = 1$ for $j = 1, \ldots, s-1$, thus

$$g_s(\tilde{c}) = \max \left\{ \begin{array}{l} \sum_{j=1}^{s-1} w_j + \sum_{j=s}^{b-1} w_j x_j : \\ \sum_{j=1}^{s-1} w_j + \sum_{j=s}^{b-1} w_j x_j \leq \tilde{c}, \\ x_j \in \{0,1\},\ j = s, \ldots, b-1. \end{array} \right. \tag{8}$$

The recursion for f_t will repeatedly insert an item to the knapsack, while the recursion for g_s will remove one item, so we get

$$f_t(\tilde{c}) = \left\{ \begin{array}{ll} f_{t-1}(\tilde{c}) & \text{for} \quad \tilde{c} = 0, \ldots, w_t - 1 \\ \max\{f_{t-1}(\tilde{c}),\ f_{t-1}(\tilde{c} - w_t) + w_t\} & \text{for} \quad \tilde{c} = w_t, \ldots, c \end{array} \right. \tag{9}$$

setting $f_{b-1}(\tilde{c}) = 0$ for $\tilde{c} = 0, \ldots, c$. The corresponding recursion for g_s becomes

$$g_s(\tilde{c}) = \left\{ \begin{array}{ll} g_{s+1}(\tilde{c}) & \text{for} \quad \tilde{c} = c - w_s + 1, \ldots, c \\ \max\{g_{s+1}(\tilde{c}),\ g_{s+1}(\tilde{c} + w_s) - w_s\} & \text{for} \quad \tilde{c} = 0, \ldots, c - w_s \end{array} \right. \tag{10}$$

with initial values $g_b(\tilde{c}) = 0$ for $\tilde{c} = 0, \ldots, c,\ \tilde{c} \neq \overline{w}$ and $g_b(\tilde{c}) = \overline{w}$ for $\tilde{c} = \overline{w}$, where $\overline{w} = \sum_{j=1}^{b-1} w_j$. Now we start from $(s,t) = (b, b-1)$ and repeatedly decrease s and increase t. At each stage we merge the two sets in $O(c)$ time, in order to find the best current solution

$$z = \max_{\tilde{c}=0,\ldots,c} f_t(\tilde{c}) + g_s(c - \tilde{c}), \tag{11}$$

and we terminate the process if $z = c$, or $(s, t) = (1, n)$ was reached.

The algorithm may be improved in those cases where $v = \sum_{j=1}^{n} w_j - \overline{w} < c$. Since there is no need for removing more items $j < b$ than can be compensated by items $j \geq b$, we may restrict recursion g_s to consider states $\tilde{c} = c - v, \ldots, c$ while recursion f_t only will consider states $\tilde{c} = 0, \ldots, v$.

The solution vector corresponding to the optimal objective value is found by backtracking through the states. To each state we associate the item number, that was added or subtracted in order to obtain the state. From the optimal choice of states $f_t(\tilde{c})$, $g_s(c - \tilde{c})$ we repeatedly subtract or add the corresponding items, obtaining new weights $\tilde{c}$ that can be found previously in the table. The process terminates when we reach $\tilde{c} = 0$ respectively $\tilde{c} = \overline{w}$.

The presented algorithm has basically the same complexity $O(nc)$ as the recursion presented by Bellman [2] but if we use a forward dynamic programming recursion, only the live states need to be considered thus leading to the complexity $O(\min\{nc,\ nv,\ 2^b + 2^{n-b}\})$. For small instances where $2^n < c$ and $b = n/2$ this improves by a square root over the Bellman recursion.

In our situation large sized instances generally have many solutions that satisfy the capacity constraint (6) with equality, and here the separability ensures a fast determination of the optimal solution. On the other hand small instances seldom meet the capacity constraint with equality thus we have to perform a complete enumeration, taking advantage of the fact that only $2^b + 2^{n-b} \ll 2^n$ states are considered.

5.1 Improvement of lower bounds

In those cases where splitting the SMKP solution succeeds, our lower bound equals the upper bound and we may immediately backtrack. Thus the performance of the MULKNAP algorithm strictly depends on whether the splitting succeeds or not.

One attempt to improve the splitting is to sort the items according to decreasing weights, before solving each Subset-sum Problem. As we use dynamic programming, this means that for each knapsack i, we choose the solution that involves the largest possible items, thus (where possible) leaving some small items for the last knapsacks.

Even with this improvement we may encounter situations where the SMKP solution cannot be split in the m knapsacks. In these situations we improve the lower bound as follows: For each knapsack i with $\sum_{j=1}^{n} w_j x_{ij} < c_i$ the chosen items are set free and a 0-1 Knapsack Problem with capacity c_i, defined on all the free items, is solved. As the items chosen for the initial Subset-sum Problem form a subset of the feasible solutions to the new problem, we may only improve the lower bound in this way.

5.2 Tightening constraints

For small instances, the upper bounds obtained by surrogate relaxation of the MKP are too optimistic, as the individual knapsacks cannot be filled as efficiently as the surrogate relaxed knapsack. To compensate for this inconvenience, we solve a series of Subset-sum Problems to tighten the capacity constraint on each knapsack.

Considering the capacity constraints in (1) for each knapsack separately, the largest possible filling $\hat{c}_i$ of knapsack i is found by solving the following Subset-sum Problem. Let $N_i = \{j > h : d_j \leq i\}$ be the set of free variables that can be assigned to knapsack i. Assuming that $N_i = \{1, \ldots, n\}$ (where n is different from the size of MKP) we get the problem:

$$\begin{aligned} \text{maximize} \quad & \hat{c}_i = \sum_{j=1}^{n} w_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} w_j x_j \leq c_i, \\ & x_j \in \{0, 1\}, \ j = 1, \ldots, n. \end{aligned} \tag{12}$$

As no optimal solution x can have weight sum $\sum_{j=1}^{n} w_j x_j > \hat{c}_i$, we may tighten the capacity constraints in (1) to

$$\sum_{j=1}^{n} w_j x_{ij} \leq \hat{c}_i, \quad i = 1, \ldots, m, \tag{13}$$

for the current branching node as well as for all descendant nodes. In this way the capacity of the surrogate relaxed problem becomes

$$c = \sum_{i=1}^{m} \hat{c}_i \leq \sum_{i=1}^{m} c_i \tag{14}$$

which leads to tighter upper bounds as well as increasing the chance for successfully splitting the solution to SMKP into the m knapsacks.

6 Reduction algorithm

As known from ordinary 0-1 Knapsack Problems, the size of a Multiple Knapsack Problem may be reduced by preprocessing [10]. For a given item j let $u_0(j)$ be any upper bound on (1) with the additional constraint $\sum_{i=1}^{m} x_{ij} = 0$. If $u_0(j) < z$ for any lower bound z, then we may add the constraint $\sum_{i=1}^{m} x_{ij} = 1$ to the problem, i.e. in every optimal solution to MKP, item j must be included in some knapsack.

In a similar way let $u_1(j)$ be any upper bound on (1) with the additional constraint $\sum_{i=1}^{m} x_{ij} = 1$. If $u_1(j) < z$, then x_{ij} may be fixed at 0 for $i = 1, \ldots, m$, thus in any optimal solution, item j cannot be included in any of the knapsacks.

Notice, that the last equation is able to fix all variables x_{ij} to their optimal value for $i = 1, \ldots, m$, while the first equation only rules out one possibility out of $m + 1$. Thus in the presented algorithm we have only incorporated the last reduction, which however is quite efficient as not only several items are removed from the problem, but generally also the capacity constraints may be tightened further when items are excluded from the problem.

Using the upper bound by Dembo and Hammer [5] we get the following reduction test: Let b be the break item of the surrogate relaxed problem, then we have the bound

$$u_1(j) = \sum_{i=1}^{b-1} p_i + p_j + \left\lfloor \left(c - \sum_{i=1}^{b-1} w_i - w_j \right) p_b / w_b \right\rfloor . \tag{15}$$

As the solution vector x corresponding to our lower bound is saved, we may use a tighter reduction than presented above, fathoming an item j whenever $u_1(j) \leq z$. This reduction is performed at each branching node of the enumeration tree, and we swap the reduced items to position $h+1$, increasing h each time. The computational complexity of the reduction is $O(n)$ for testing n items which takes a negligible effort compared to the determination of upper and lower bounds.

7 Computational experiments

The presented MULKNAP algorithm has been implemented in C and the source code is available from `http://www.diku.dk/~pisinger`. We will compare MULKNAP with the MTM algorithm by Martello and Toth for several groups of randomly generated instances. The code for MTM has been obtained from [10], and all tests have been run on a HP9000/730. We consider four different types of randomly generated data instances, for different ranges $R = 100,\ 1000$ and $10\,000$. The instances are generated as described in [10], having four classes of instances.

Uncorrelated data instances: p_j and w_j are randomly distributed in $[10, R]$. *Weakly correlated data instances*: w_j is randomly distributed in $[10, R]$ and p_j is randomly distributed in $[w_j - R/10, w_j + R/10]$ such that $p_j \geq 1$. *Strongly correlated data instances*: w_j is randomly distributed in $[10, R]$ and p_j is set to $w_j + 10$. *Subset-sum data instances*: w_j randomly distributed in $[10, R]$ and p_j equals w_j.

Two different classes of capacities are considered: *Similar capacities* have the first $m-1$ capacities c_i randomly distributed in

$$\left[\; 0.4 \sum_{j=1}^{n} w_j / m, \;\; 0.6 \sum_{j=1}^{n} w_j / m \; \right] \text{ for } i = 1, \ldots, m-1, \tag{16}$$

while *dissimilar capacities* have c_i distributed in

$$\left[\; 0, \;\; 0.5 \left(\sum_{j=1}^{n} w_j - \sum_{k=1}^{i-1} c_k \right) \; \right] \text{ for } i = 1, \ldots, m-1. \tag{17}$$

The last capacity c_m is in both classes chosen as $c_m = 0.5 \sum_{j=1}^{n} w_j - \sum_{i=1}^{m-1} c_i$, to ensure that the sum of the capacities is half of the total weight sum. For each instance we check whether the assumptions (2) are respected, and generate a new instance if a constraint is violated.

A maximum amount of 3 hours was given to each algorithm for solving the 20 instances in each class, since this means that every instance is solved in about 10 minutes, what will be considered "reasonable time". A dash in the following tables indicates that the 20 problems could not be solved within this time limit. The MTM algorithm, which originally is designed for problems up to $n = 1000$, was extended such that problems up

Table 1: Tightening of constraints, small problems with $m = 5$. Max. of 20 instances.

$n \setminus R$	Uncorrelated 100	Uncorrelated 1000	Uncorrelated 10 000	Weakly correlated 100	Weakly correlated 1000	Weakly correlated 10 000	Strongly correlated 100	Strongly correlated 1000	Strongly correlated 10 000	Subset-sum 100	Subset-sum 1000	Subset-sum 10 000
25	158	1159	14989	164	2057	15952	63	499	4372	0	380	3671
50	0	0	0	0	0	19881	0	0	0	0	0	0
75	0	0	0	0	0	0	0	0	0	0	0	0
100	0	0	0	0	0	0	0	0	0	0	0	0
200	0	0	0	0	0	0	0	0	0	0	0	0
25	35	692	3747	131	1238	11010	37	260	2865	14	204	2865
50	5	34	887	11	811	5677	2	21	486	2	30	540
75	4	30	310	12	92	1244	1	48	268	1	48	524
100	4	5	83	16	101	2305	1	4	218	1	4	207
200	2	3	11	2	0	22	1	1	20	1	1	50
	Top: similar capacities. Bottom: dissimilar capacities.											

to $n = 30\,000$ could be solved. The MULKNAP algorithm — demanding less static space — was run with instances up to $n = 100\,000$. Small data instances (i.e. $n \leq 200$) were tested with $m = 5$ knapsacks, while large instances (i.e. $n \geq 300$) were tested with $m = 10$. None of the algorithms were able to solve instances with small values of n/m. In each of the following tables, instances with similar capacities are considered in the upper part of the table, while the lower part of the table considers instances of dissimilar capacities.

Tables 1 to 5 consider small sized data instances. First, we characterize the individual parts of MULKNAP. Table 1 shows the efficiency of the tightening of the constraints. For each instance we report the largest tightening $\sum_{i=1}^{m}(c_i - \hat{c}_i)$, where c_i are the original capacities and $\hat{c}_i$ are the tightened capacities. It is seen that for small instances with large range R the tightening is quite efficient, significantly contributing to the bounding process. For problems larger than $n = 100$ there is little gain of the tightening approach, thus the tightening may be omitted for e.g. $n > 300$.

Table 2 reports the largest number of reduced items at any stage of the branch-and-bound process. It is seen that especially the weakly correlated problems may be reduced significantly, but also some uncorrelated instances can be slightly reduced. The entries

Table 2: Number of items reduced, small problems with $m = 5$. Max. of 20 instances.

$n \setminus R$	Uncorrelated 100	Uncorrelated 1000	Uncorrelated 10 000	Weakly correlated 100	Weakly correlated 1000	Weakly correlated 10 000	Strongly correlated 100	Strongly correlated 1000	Strongly correlated 10 000	Subset-sum 100	Subset-sum 1000	Subset-sum 10 000
25	11	10	10	13	13	13	9	6	2	0	0	0
50	0	0	0	0	0	23	0	0	0	0	0	0
75	0	0	0	0	0	0	0	0	0	0	0	0
100	0	0	0	0	0	0	0	0	0	0	0	0
200	0	0	0	0	0	0	0	0	0	0	0	0
25	10	8	7	13	12	12	1	2	0	0	0	0
50	0	0	0	19	25	20	0	0	0	0	0	0
75	0	23	25	32	30	32	0	0	0	0	0	0
100	0	0	0	43	44	46	0	0	0	0	0	0
200	0	0	0	0	0	0	0	0	0	0	0	0
	Top: similar capacities. Bottom: dissimilar capacities.											

Table 3: Number of recursive calls to MULBRANCH, small problems with $m = 5$. Avrg. of 20 instances.

	Uncorrelated			Weakly correlated			Strongly correlated			Subset-sum		
$n \setminus R$	100	1000	10 000	100	1000	10 000	100	1000	10 000	100	1000	10 000
25	1318	1067	664	8902	17912	9382	122	1174	670	1	517	595
50	1	1	1	1	1	3985	1	1	1	1	1	1
75	1	1	1	1	1	1	1	1	1	1	1	1
100	1	1	1	1	1	1	1	1	1	1	1	1
200	1	1	1	1	1	1	1	1	1	1	1	1
25	8	170	44	359	250	1498	2	57	62	2	27	43
50	1	1	1	3	212	1018	1	2	7	1	3	7
75	1	1	1	2	6	37	1	3	4	1	2	4
100	1	1	1	3	8	54	1	1	2	1	1	2
200	1	1	1	1	1	1	1	1	1	1	1	2
	Top: similar capacities. Bottom: dissimilar capacities.											

Table 4: Computing time MULKNAP, small problems with $m = 5$. Avrg. of 20 instances.

	Uncorrelated			Weakly correlated			Strongly correlated			Subset-sum		
$n \setminus R$	100	1000	10 000	100	1000	10 000	100	1000	10 000	100	1000	10 000
25	0.33	0.41	0.27	3.24	10.00	6.33	0.06	1.76	2.92	0.00	1.35	10.77
50	0.00	0.00	0.01	0.00	0.00	11.87	0.00	0.01	0.10	0.00	0.00	0.08
75	0.00	0.00	0.01	0.00	0.00	0.01	0.00	0.02	0.18	0.00	0.00	0.04
100	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.02	0.20	0.00	0.00	0.04
200	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.08	0.98	0.00	0.00	0.03
25	0.00	0.07	0.02	0.12	0.14	0.62	0.00	0.10	0.36	0.00	0.09	1.06
50	0.00	0.00	0.01	0.00	0.23	2.06	0.00	0.02	0.55	0.00	0.01	0.32
75	0.00	0.00	0.01	0.00	0.01	0.07	0.01	0.06	1.38	0.00	0.01	0.23
100	0.00	0.00	0.01	0.00	0.02	0.18	0.01	0.02	0.35	0.00	0.01	0.13
200	0.00	0.00	0.01	0.00	0.01	0.01	0.01	0.08	1.14	0.00	0.00	0.10
	Top: similar capacities. Bottom: dissimilar capacities.											

Table 5: Computing time MTM, small problems with $m = 5$. Avrg. of 20 instances.

	Uncorrelated			Weakly correlated			Strongly correlated			Subset-sum		
$n \setminus R$	100	1000	10 000	100	1000	10 000	100	1000	10 000	100	1000	10 000
25	0.43	1.03	0.96	5.11	10.18	5.69	0.48	12.43	37.09	0.00	25.21	143.31
50	0.08	2.75	8.02	0.82	80.61	234.49	3.86	8.54	—	0.00	0.01	0.07
75	0.06	1.52	17.36	0.53	27.17	158.98	—	162.26	412.30	0.00	0.01	0.06
100	0.07	1.57	13.13	0.13	27.15	309.48	—	—	—	0.00	0.01	0.07
200	0.04	1.60	7.31	0.02	20.97	358.50	—	—	—	0.00	0.01	0.06
25	0.01	0.04	0.04	0.14	0.11	0.63	0.02	0.29	1.20	0.00	0.53	4.48
50	0.01	0.11	0.87	0.15	2.25	18.16	0.13	2.07	122.47	0.00	0.22	11.55
75	0.01	0.24	0.93	0.02	2.83	17.21	7.02	89.90	136.46	0.00	0.06	3.99
100	0.00	0.17	2.02	0.02	3.30	17.19	—	157.60	169.60	0.00	0.01	0.98
200	0.01	0.07	0.62	0.01	9.20	122.04	—	—	—	0.00	0.01	0.33
	Top: similar capacities. Bottom: dissimilar capacities.											

Table 6: Computing time MULKNAP, large problems with $m = 10$. Avrg. of 20 instances.

	Uncorrelated			Weakly correlated			Strongly correlated			Subset-sum		
$n \setminus R$	100	1000	10 000	100	1000	10 000	100	1000	10 000	100	1000	10 000
100	0.00	0.00	0.03	0.00	0.01	0.03	0.01	0.02	0.22	0.00	0.01	0.08
300	0.00	0.01	0.01	0.00	0.01	0.02	0.01	0.15	1.15	0.00	0.01	0.05
1000	0.00	0.00	0.00	0.00	0.00	0.01	0.06	0.60	5.83	0.00	0.01	0.04
3000	0.00	0.01	0.01	0.00	0.01	0.02	0.15	1.98	20.88	0.01	0.01	0.04
10000	0.01	0.02	0.03	0.01	0.01	0.04	0.85	7.29	135.36	0.02	0.02	0.05
30000	0.04	0.04	0.07	0.04	0.04	0.07	3.07	33.39	491.87	0.05	0.05	0.08
100000	0.18	0.19	0.24	0.17	0.19	0.22	10.91	149.09	1176.09	0.22	0.20	0.29
100	0.02	0.00	0.60	0.01	0.38	19.78	0.00	1.96	12.80	0.00	0.26	8.55
300	0.00	0.01	0.05	0.00	0.01	0.20	0.02	3.16	4.04	0.00	0.14	0.35
1000	0.01	0.01	0.02	0.01	0.03	0.03	0.07	0.48	46.30	0.00	0.01	0.14
3000	0.00	0.01	0.02	0.01	0.01	0.08	0.17	2.04	27.45	0.01	0.01	0.08
10000	0.01	0.02	0.04	0.01	0.02	0.05	0.63	6.92	144.33	0.02	0.03	0.07
30000	0.04	0.04	0.09	0.04	0.04	0.08	2.71	33.09	426.42	0.05	0.06	0.09
100000	0.18	0.19	0.25	0.18	0.19	0.22	12.61	147.46	1162.63	0.22	0.20	0.30
	Top: similar capacities. Bottom: dissimilar capacities.											

should however be read with care, as problems that are solved during the first iteration of the algorithm, never will reduce any items, while hard problems with a large search tree generally will reduce a considerable amount of items. Thus the table is only presented to show the effect in branching intensive situations.

The enumerative hardness of each instance is given in Table 3, where we measure the number of recursive procedure calls made by MULBRANCH. It is seen that most of the large problems are solved in the first iteration, while especially problems around $n = 25$ items demand a very extensive search. It seems that large ranged problems demand more enumeration, as it is more difficult to split the solution to the surrogate relaxed problem into the m knapsacks.

Table 7: Computing time MTM, large problems with $m = 10$. Avrg. of 20 instances.

	Uncorrelated			Weakly correlated			Strongly correlated			Subset-sum		
$n \setminus R$	100	1000	10 000	100	1000	10 000	100	1000	10 000	100	1000	10 000
100	1.72	338.81	—	53.88	—	—	—	—	—	0.00	0.01	0.08
300	0.12	31.46	501.33	0.12	405.04	—	—	—	—	0.00	0.01	0.10
1000	0.03	39.78	—	0.01	114.20	—	—	—	—	0.01	0.02	0.12
3000	0.07	16.12	—	0.06	15.01	—	—	—	—	0.05	0.07	0.15
10000	0.97	9.53	—	0.41	3.01	—	—	—	—	0.34	0.35	0.45
30000	35.05	1.20	—	1.11	1.23	—	—	—	—	1.09	1.10	1.30
100	0.15	0.94	14.79	0.58	45.28	—	435.22	—	—	0.00	1.61	412.66
300	0.06	1.96	24.18	0.06	116.63	—	—	—	—	0.00	0.20	4.89
1000	0.10	2.37	77.40	0.03	21.69	—	—	—	—	0.01	0.04	3.97
3000	0.40	2.95	202.97	0.07	2.49	—	—	—	—	0.06	0.07	0.33
10000	4.82	2.99	255.35	0.56	0.99	—	—	—	—	0.39	0.38	0.46
30000	—	2.03	411.66	207.32	1.23	161.06	—	—	—	1.26	1.20	1.37
	Top: similar capacities. Bottom: dissimilar capacities.											

Table 4 and 5 finally compare the solution times of MULKNAP with those of MTM. It is seen that MULKNAP is able to solve all the instances in reasonable time, while MTM has considerable problems for large sized and large ranged problems of weak and strong correlation. If the individual entries are compared, it is seen that MULKNAP generally has faster solution times than MTM, and the larger the problems become, the more efficient MULKNAP gets.

The last two tables, Table 6 and 7 compare the solution times of the two algorithms for large instances $n \geq 100$. It is seen that MULKNAP is able to solve all of the large sized instances while MTM only can solve low-ranged problems. For very large instances $n \geq 1000$, MULKNAP is actually able to solve the problems in times comparable to the best solution times for the 0-1 Knapsack Problem.

8 Conclusion

The presented MULKNAP algorithm is based on the MTM framework by Martello and Toth [8], but lower bounds are derived from the surrogate solution by solving a series of Subset-sum Problems, and a specialized algorithm is used for tightening the capacity constraints. Moreover efficient algorithms are used for deriving upper bounds as well as for solving the Subset-sum Problem. In this way MULKNAP is the first algorithm to solve all large sized instances up to $n = 100\,000$ with data range up to $R = 10\,000$. It is also the first algorithm to solve strongly correlated instances of large size.

Although the MKP is $\mathcal{NP}$-hard in the strong sense, and thus is badly suited for dynamic programming, the present work has shown that dynamic programming algorithms may be used to solve specific subproblems with great success. The tightening of capacities would not have been possible without a pseudopolynomial time bound, as any branch-and-bound algorithm would use exponential time in those cases where the capacity bound may be tightened. Also since we solve the 0-1 Knapsack Problems through dynamic programming, reasonable times are seen even for strongly correlated problems.

References

[1] E. Balas and E. Zemel (1980), "An Algorithm for Large Zero-One Knapsack Problems", *Operations Research*, **28**, 1130–1154.

[2] R. E. Bellman (1957), *Dynamic programming*, Princeton University Press, Princeton, NJ.

[3] N. Christofides, A. Mingozzi, P. Toth (1979). In N. Christofides, A. Mingozzi, P. Toth, C. Sandi (eds.), *Combinatorial Optimization*, Wiley, Chichester, 339–369.

[4] G. B. Dantzig (1957), "Discrete Variable Extremum Problems", *Operations Research*, **5**, 266–277.

[5] R. S. Dembo and P. L. Hammer (1980), "A Reduction Algorithm for Knapsack Problems", *Methods of Operations Research*, **36**, 49–60.

[6] E. Horowitz and S. Sahni (1974), "Computing partitions with applications to the Knapsack Problem", *Journal of ACM*, **21**, 277–292.

[7] M. S. Hung and J. C. Fisk (1978), "An algorithm for 0-1 multiple knapsack problems", *Naval Research Logistics Quarterly*, **24**, 571–579.

[8] S. Martello and P. Toth (1980), "Solution of the zero-one multiple knapsack problem", *European Journal of Operational Research*, **4**, 276–283.

[9] S. Martello and P. Toth (1981), "A bound and bound algorithm for the zero-one multiple knapsack problem", *Discrete Applied Mathematics*, **3**, 275–288.

[10] S. Martello and P. Toth (1990), *Knapsack Problems: Algorithms and Computer Implementations,* Wiley, Chichester, England.

[11] A. Neebe and D. Dannenbring (1977), "Algorithms for a specialized segregated storage problem", *University of North Carolina*, Technical Report 77-5.

[12] D. Pisinger (1995), "An $O(nr)$ Algorithm for the Subset Sum Problem", *DIKU, University of Copenhagen, Denmark,* Report 95/1.

[13] D. Pisinger (1997), "A minimal algorithm for the 0-1 knapsack problem", *Operations Research*, **45**, 758–767

[14] D. Pisinger (1995), "The Multiple Loading Problem", *Proceedings NOAS'95*, University of Reykjavík, Iceland, August 18–19, 1995.

[15] G. T. Ross and R. M. Soland (1975), "A branch and bound algorithm for the generalized assignment problem", *Mathematical Programming*, **8**, 91–103.

[16] Z. Sinuany-Stern and I. Winer (1994), "The one dimensional cutting stock problem using two objectives", *Journal of the Operational Research Society*, **45**, 231–236.