ELSEVIER

European Journal of Operational Research 83 (1995) 394–410

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

A minimal algorithm for the Multiple-Choice Knapsack Problem

David Pisinger

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

Abstract

The Multiple-Choice Knapsack Problem is defined as a 0–1 Knapsack Problem with the addition of disjoined multiple-choice constraints. As for other knapsack problems most of the computational effort in the solution of these problems is used for sorting and reduction. But although $O(n)$ algorithms which solve the linear Multiple-Choice Knapsack Problem without sorting have been known for more than a decade, such techniques have not been used in enumerative algorithms. In this paper we present a simple $O(n)$ partitioning algorithm for deriving the optimal linear solution, and show how it may be incorporated in a dynamic programming algorithm such that a minimal number of classes are enumerated, sorted and reduced. Computational experiments indicate that this approach leads to a very efficient algorithm which outperforms any known algorithm for the problem.

Keywords: Knapsack problem; Dynamic programming; Reduction

1. Introduction

Given k classes $N_1, \ldots, N_k$ of *items* to pack in some knapsack of *capacity* c. Each item $j \in N_i$ has a *profit* p_{ij} and a *weight* w_{ij}, and the problem is to choose one item from each class such that the profit sum is maximized without having the weight sum to exceed c. The *Multiple-Choice Knapsack Problem* (MCKP) may thus be formulated as:

$$\text{maximize} \quad z = \sum_{i=1}^{k} \sum_{j \in N_i} p_{ij} x_{ij} \tag{1}$$

$$\text{subject to} \quad \sum_{i=1}^{k} \sum_{j \in N_i} w_{ij} x_{ij} \leq c,$$

$$\sum_{j \in N_i} x_{ij} = 1, \quad i = 1, \ldots, k,$$

$$x_{ij} \in \{0, 1\}, \quad i = 1, \ldots, k, \quad j \in N_i.$$

All coefficients p_{ij}, w_{ij}, and c are positive integers, and the classes $N_1, \ldots, N_k$ are mutually disjoint, class N_i having *size* n_i. The *total* number of items is $n = \sum_{i=1}^{k} n_i$.

SSDI 0377-2217(95)00015-1

Negative coefficients p_{ij}, w_{ij} in (1) may be handled by adding a sufficiently large constant to all items in the corresponding class as well as to c. To avoid unsolvable or trivial situations we assume that

$$\sum_{i=1}^{k} \min_{j \in N_i} w_{ij} \leq c < \sum_{i=1}^{k} \max_{j \in N_i} w_{ij}. \tag{2}$$

If we relax the integrality constraint $x_{ij} \in \{0, 1\}$ in (1) to $0 \leq x_{ij} \leq 1$ we obtain the *Linear Multiple-Choice Knapsack Problem* (LMCKP). If each class has two items, where $(p_{i1}, w_{i1}) = (0, 0)$, $i = 1, \ldots, k$, the problem (1) corresponds to the *0–1 Knapsack Problem* (KP). The linear version of KP will be denoted by LKP.

MCKP is NP-hard as it contains KP as a special case, but it can be solved in pseudo-polynomial time through dynamic programming (Dudzinski and Walukiewicz, 1987). The problem has a large range of applications: Capital Budgeting (Nauss, 1978), Menu Planning (Sinha and Zoltners, 1979), transforming nonlinear KP to MCKP (Nauss, 1978), determining which components should be linked in series in order to maximize fault tolerance (Sinha and Zoltners, 1979), and to accelerate ordinary LP/GUB problems by the dual simplex algorithm (Witzgal, 1977). Moreover MCKP appear by Lagrange relaxation of several integer programming problems (Fisher, 1981).

Several algorithms for MCKP have been presented during the last two decades: e.g. Nauss (1978), Sinha and Zoltners (1979), and Dyer, Kayal and Walker (1984). Most of these algorithms start by solving LMCKP in order to obtain an upper bound for the problem. LMCKP is solved in two steps: 1) The LP-dominated items are reduced by sorting the items in each class according to nondecreasing weights, and then applying some dominance criteria to delete unpromising states; 2) the reduced LMCKP is solved by a greedy algorithm. After these two initial steps, upper bound tests may be used to fix several variables in each class to their optimal value. The reduced MCKP problem is then solved to optimality through enumeration (Dudzinski and Walukiewicz, 1987).

Essential results in the field of KP however indicates that MCKP may be solved easier: Balas and Zemel (1980) independently with Fayard and Plateau (1977) proposed to consider only a small subset of the items – the so-called *core* – in order to solve KP. A core can be found in $O(n)$ time through a partitioning algorithm, and since the restricted KP defined on the core items is easy to solve for several classes of data instances, it means that many instances may be solved in linear time (Martello and Toth, 1988; Pisinger, 1995a). However if optimality of the core solution cannot be proved, a complete enumeration including the variables outside the core has to be performed.

Although $O(n)$ algorithms for LMCKP have been known for a decade (Zemel, 1984; Dyer, 1984), making it possible to derive a 'core' reasonably easy, a similar technique has not been used for MCKP. According to Martello and Toth (1990) this may be caused by the fact that reduction of LP-dominated items is necessary in order to derive upper bounds in a branch-and-bound algorithm. The current paper however demonstrates that a core algorithm for MCKP is possible, although several questions had to be answered: Which items or classes should be included in the core? How should we derive upper bounds in a branch-and-bound algorithm when LP-dominated items have not been deleted? How should a core be derived? How should a primal algorithm for LMCKP be developed?

The present paper is a counterpart to a minimal algorithm for KP by Pisinger (1995c): A simple algorithm is used for solving LMCKP, and for deriving an initial feasible solution to MCKP. Starting from this initial solution we use dynamic programming to solve MCKP, adding new classes to the core by need. By this technique we are able to show that a minimal number of classes are considered in order to solve MCKP to optimality.

The paper is organized in the following way: First, Section 2 brings some basic definitions, and shows fundamental properties of MCKP, while Section 3 presents a simple partitioning algorithm for the solution of LMCKP. Next, Section 4 shows how gradients may be used in an expanding-core, as well as

presenting some logical tests which may be used to fix variables at their optimal value, before a class is added to the core. Section 5 gives a description of the dynamic programming algorithm, and Section 6 shows how we keep track on the solution vector in dynamic programming. Finally Section 7 gives the main algorithm and proves minimality, while Section 8 brings computational experiments.

2. Fundamental properties

Definition 1. If two items r and s in the same class N_i satisfy that

$$w_{ir} \leq w_{is} \quad \text{and} \quad p_{ir} \geq p_{is}, \tag{3}$$

then we say that item r *dominates* item s. Similarly if some items $r, s, t \in N_i$ with $w_{ir} \leq w_{is} \leq w_{it}$ and $p_{ir} \leq p_{is} \leq p_{it}$ satisfy

$$\det(w_{is} - w_{ir},\ p_{is} - p_{ir},\ w_{it} - w_{ir},\ p_{it} - p_{ir}) \leq 0, \tag{4}$$

then we say that item s is *LP-dominated* by items r and t.

Proposition 1 (Sinha and Zoltners, 1979). *Given two items $r, s \in N_i$. If item r dominates item s then an optimal solution to* MCKP *with $x_{is} = 0$ exists. If two items $r, t \in N_i$ LP-dominate an item $s \in N_i$ then an optimal solution to* LMCKP *with $x_{is} = 0$ exists.*

As a consequence, we only have to consider *LP-undominated* items R_i in the solution of LMCKP. Note that these items form the upper convex boundary of the set N_i, as illustrated in Fig. 1. The set of LP-undominated items may be found by ordering the items in each class N_i according to increasing weights, and successively test the items according to criteria (3) and (4). If two items have the same weight and profit, choose an arbitrary of them. Now LMCKP may be solved by using the *greedy algorithm*:

Algorithm 1. *Greedy.*

Step 1. Find the LP-undominated classes R_i (ordered by increasing weights) for all classes N_i, $i = 1, \ldots, k$. Choose the lightest item from each class (i.e. set $x_{i1} = 1$, $x_{ij} = 0$ for $j = 2, \ldots, |R_i|$, $i = 1, \ldots, k$)

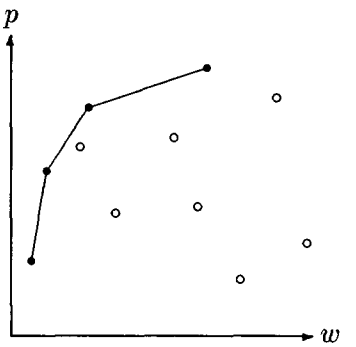

Fig. 1. LP-undominated items R_i (black) form the upper convex boundary of N_i.

and define the chosen weight and profit sum as $W = \sum_{i=1}^{k} w_{i1}$, resp. $P = \sum_{i=1}^{k} p_{i1}$. For all items $j \neq 1$ define the slope λ_{ij} as

$$\lambda_{ij} = \frac{p_{ij} - p_{i,j-1}}{w_{ij} - w_{i,j-1}}, \quad i = 1,\ldots,k, \quad j = 2,\ldots,|R_i|. \tag{5}$$

This slope is a measure of the profit-to-weight ratio obtained by choosing item j instead of item $j-1$ in class R_i (Zemel, 1980). Using the greedy principle, order the slopes $\{\lambda_{ij}\}$ in nondecreasing order.

Step 2. Let i, j be the indices corresponding to the next slope λ_{ij} in $\{\lambda_{ij}\}$. If $W + w_{ij} > c$, goto Step 3. Otherwise set $x_{ij} = 1$, $x_{i,j-1} = 0$ and update the sums $W = W + w_{ij} - w_{i,j-1}$, $P = P + p_{ij} - p_{i,j-1}$. Repeat Step 2.

Step 3. If $W = c$ we have an integer solution and the optimal objective value to LMCKP (and MCKP) is $z^* = P$. Otherwise let λ_{ij} be the next slope in the list. We have two fractional variables $x_{ij} = (c - W)/(w_{ij} - w_{i,j-1})$ respectively $x_{i,j-1} = 1 - x_{ij}$, which both belong to the same class. The optimal objective value is

$$z^* = P + (c - W)\lambda_{ij}. \tag{6}$$

Although several orderings of $\{\lambda_{ij}\}$ exist in Step 1 when more items have the same slope, we will assume that one specific ordering has been chosen.

The *LP-optimal choices* b_i obtained by Algorithm 1 are those variables, where $x_{ib_i} = 1$. The class containing two fractional variables in Step 3 will be denoted the *fractional class* N_a, and the *fractional variables* are x_{ab_a}, $x_{ab'_a}$ possibly with $x_{ab'_a} = 0$. An initial feasible solution to MCKP may be constructed by choosing the LP-optimal variables, i.e. setting $x_{ib_i} = 1$ for $i = 1,\ldots,k$ and $x_{ij} = 0$ for $i = 1,\ldots,k$, $j \neq b_i$. The solution will be denoted the *break solution* and the corresponding weight and profit sum is W resp. P.

Proposition 2. *As a consequence of Algorithm* 1 *an optimal solution* x^* *to* LMCKP *satisfies the following*: 1) x^* *has at most two fractional variables* x_{ab_a} *and* $x_{ab'_a}$; 2) *If* x^* *has two fractional variables they must be adjacent variables within the same class* N_a; 3) *If* x^* *has no fractional variables then the break solution is an optimal solution to* MCKP.

The presented greedy algorithm has time complexity $O(n \log n)$ due to the ordering of slopes. It should be mentioned, that when the classes form a KP, Algorithm 1 is exactly the *greedy algorithm* for LKP, and the objective value (6) corresponds to the *Dantzig upper bound* for KP (Dantzig, 1957).

An optimal solution to MCKP generally corresponds to the break solution, except for some few classes where other items than the LP-optimal choices have been chosen. This property may be illustrated the following way: Define the positive and negative gradient λ_i^+ and λ_i^- for each class N_i, $i \neq a$ as (see Fig. 2)

$$\lambda_i^+ = \max_{j \in N_i, w_{ij} > w_{ib_i}} (p_{ij} - p_{ib_i})/(w_{ij} - w_{ib_i}), \quad i = 1,\ldots,k, \ i \neq a, \tag{7}$$

$$\lambda_i^- = \min_{j \in N_i, w_{ij} < w_{ib_i}} (p_{ib_i} - p_{ij})/(w_{ib_i} - w_{ij}), \quad i = 1,\ldots,k, \ i \neq a, \tag{8}$$

and we set $\lambda_i^+ = 0$ (resp. $\lambda_i^- = \infty$) if the set we are maximizing (resp. minimizing) over is empty. Note that the above definitions do not demand any preprocessing of the items. The gradients are a measure of the expected gain (resp. loss) per weight unit by choosing a heavier (resp. lighter) item from N_i instead of the LP-optimal choice b_i. The gradient of the fractional class N_a is defined as

$$\lambda = (p_{ab'_a} - p_{ab_a})/(w_{ab'_a} - w_{ab_a}). \tag{9}$$

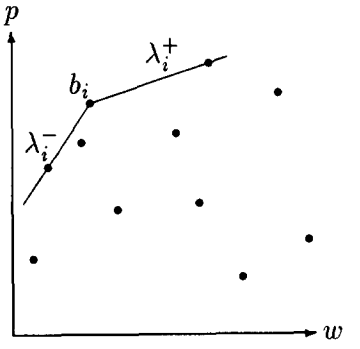

Fig. 2. Gradients λ_i^+, λ_i^- in class N_i.

In Fig. 3 we have ordered the classes according to decreasing λ_i^+ and show how often the IP-optimal solution to MCKP differs from the LP-optimal choice in each class N_i. The figure is a result of 5000 randomly generated data instances ($k = 100$, $n_i = 10$), where we have measured how often the IP-optimal choice j (satisfying $w_{ij} > w_{ib_i}$ since we are considering forward gradients) differs from the LP-optimal choice b_i in each class N_i. It is seen, that when λ_i^+ is decreasing, so is the probability that b_i is not the IP-optimal choice. Similarly in Fig. 4 we have ordered the classes according to increasing λ_i^- to show how the probability for changes decreases with increased λ_i^-.

This observation motivates considering only a small number of the classes N_i, namely those classes where λ_i^+ or λ_i^- are sufficiently close to λ. Thus at any stage the *core* is simply a set of classes $\{N_{r_1}, \ldots, N_{r_m}\}$ where $r_1, \ldots, r_m \in \{1, \ldots, k\}$. Initially the core consists of the break set N_a and we expand the core by need; alternately including the next unused class N_i which has largest λ_i^+ or smallest λ_i^-.

Since a complete enumeration of the core demands considering up to $n_{r_1} \cdot n_{r_2} \cdot \cdots \cdot n_{r_m}$ states, care should be taken before including a new class to the core. We use an upper bound test to fix as many variables at their optimal value as possible in the class before it is included in the core. If only one item remains, the class may be fathomed. Otherwise we order the remaining variables by nondecreasing weight and use test (3) to delete dominated items. The remaining class is added to the core and the new choices are enumerated through dynamic programming.

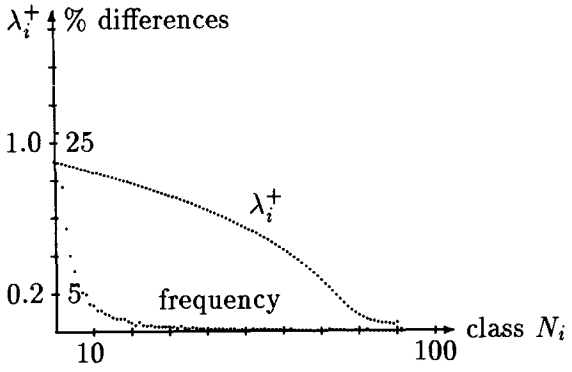

Fig. 3. Frequency of classes N_i where IP-optimal choice differs from LP-optimal choice, compared to gradient λ_i^+.

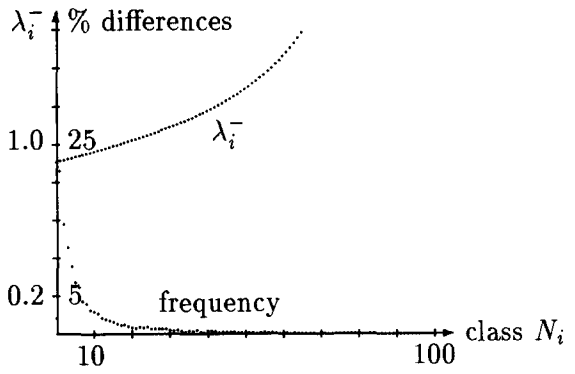

Fig. 4. Frequency of classes N_i where IP-optimal choice differs from LP-optimal choice, compared to gradient λ_i^-.

3. A partitioning algorithm for the LMCKP

Dyer (1984) and Zemel (1984) independently of each other developed O(n) algorithms for LMCKP. Both algorithms are based on the convexity of the LP-dual problem to (1), which makes it possible to *pair* the dual line segments, so that at each iteration at least 1/6 of the line segments are deleted. When the classes form a KP the algorithms reduce to that of Balas and Zemel (1980) for LKP. As Martello and Toth (1988) modified the Balas and Zemel algorithm for LKP to a primal approach which is easier to implement, we will now modify the Dyer and Zemel algorithm for LMCKP in a similar way.

Assume that N_a is the fractional class and that items b_a and b'_a are the fractional variables in N_a, such that $x_{ab_a} + x_{ab'_a} = 1$, possibly with $x_{ab'_a} = 0$. Moreover let b_i be the LP-optimal choice in class N_i, $i = 1, \ldots, k$, $i \neq a$. Due to the properties of LMCKP given in Proposition 2, LMCKP may be reformulated as finding the slope

$$\lambda = (\delta\bar{p}/\delta\bar{w}) = (p_{ab'_a} - p_{ab_a})/(w_{ab'_a} - w_{ab_a}), \tag{10}$$

such that the weight sum of the LP-optimal choices satisfy

$$\sum_{i \neq a} w_{ib_i} + w_{ab_a} \leq c < \sum_{i \neq a} w_{ib_i} + w_{ab'_a}, \tag{11}$$

$$\det(w_{ij}, p_{ij}, \delta\bar{w}, \delta\bar{p}) \leq \det(w_{ib_i}, p_{ib_i}, \delta\bar{w}, \delta\bar{p}), \quad i = 1, \ldots, k, \quad j = 1, \ldots, n_i. \tag{12}$$

Here (11) ensures that N_a is the fractional class, and (12) ensures that each item $b_i \in N_i$ is at the upper convex boundary of the set.

The formulation (10)–(12) allows us to use a partitioning algorithm for finding the optimal slope λ. In the following algorithm we assume that the classes of items N_i are represented as a list $[N_1, \ldots, N_k]$ and items in each class are also represented as a list $[j_1, \ldots, j_{n_i}]$. Elements may be deleted from a list by swapping the deleted element to the end of the list, and subsequently decreasing the list's length. Thus at any step, k and n_i refer to the current number of elements in the list. The partitioning algorithm looks like this:

Algorithm 2. *Partition.*

Step 0. Preprocess. For all classes $i = 1, \ldots, k$ let α_i and β_i be indices to the items having minimal weight (resp. maximal profit) in N_i (see Fig. 5). In case of several items satisfying the criterion, choose the item having largest profit for α_i and smallest weight for β_i. Set $W = P = 0$, and remove those items $j \neq \beta_i$ which have $w_{ij} \geq w_{i\beta_i}$ and $p_{ij} \leq p_{i\beta_i}$, since these are dominated by item β_i. If the

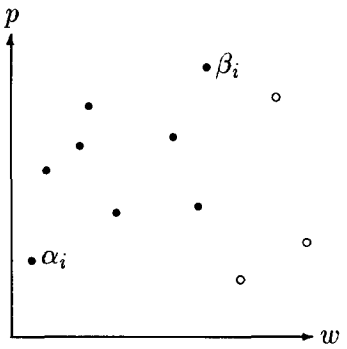

Fig. 5. Preprocessing of N_i. White nodes are dominated by β_i.

class N_i has only one item left, save the LP-optimal choice $b_i = \beta_i$ and set $W = W + w_{ib_i}$, $P = P + p_{ib_i}$, then delete class N_i.

Step 1. Choose median. For M randomly chosen classes N_i define the corresponding slope $\lambda_i = (\delta p_i / \delta w_i) = (p_{i\beta_i} - p_{i\alpha_i})/(w_{i\beta_i} - w_{i\alpha_i})$. Let $\lambda = (\delta\bar{p}/\delta\bar{w})$ be the median of these M slopes.

Step 2. Find the conclusion. For each class N_i find the items which maximize the projection on the normal to $(\delta\bar{w}, \delta\bar{p})$, i.e. which maximize the determinant

$$\det(w_{ij}, p_{ij}, \delta\bar{w}, \delta\bar{p}) = w_{ij}\delta\bar{p} - p_{ij}\delta\bar{w}. \tag{13}$$

See Fig. 6. We swap these items to the beginning of the list such that they have indices $\{1, \ldots, \ell_i\}$ in class N_i.

Step 3. Determine weight sum of conclusion. Let g_i, h_i be the lightest (resp. heaviest) item among $\{1, \ldots, \ell_i\}$ in class N_i, and let W' and W'' be the corresponding weight sums. Thus $W' = W + \sum_{i=1}^{k} w_{ig_i}$ and $W'' = W + \sum_{i=1}^{k} w_{ih_i}$.

Step 4. Check for optimal partitioning. If $W' \le c \le W''$ the partitioning at $(\delta\bar{w}, \delta\bar{p})$ is optimal. First, choose the lightest items from each class by setting $b_i = g_i$, $W = W + w_{ib_i}$, $P = P + p_{ib_i}$. Then while $W - w_{ig_i} + w_{ih_i} \le c$ run through the classes where $\ell_i \ne 1$ and choose the heaviest item by setting $b_i = h_i$, $W = W - w_{ig_i} + w_{ib_i}$, $P = P - p_{ig_i} + p_{ib_i}$. The first class where $W - w_{ig_i} + w_{ih_i} > c$ is the fractional class N_a and an optimal objective value to LMCKP is $z_{\text{LMCKP}} = P + (c - W)\lambda$. If no fractional class is defined, the LP-solution is also the optimal IP-solution. Stop.

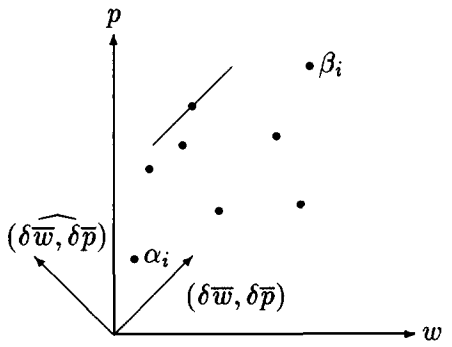

Fig. 6. Conclusion of N_i.

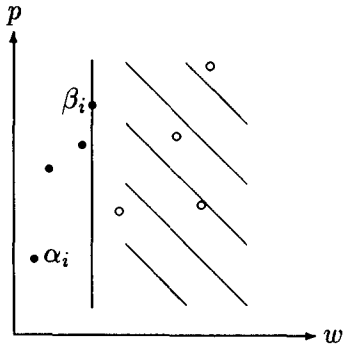

Fig. 7. Partition set N_i.

Step 5. Partition. We have one of the following two cases: 1) If $W' > c$ then the slope λ was too small (see Fig. 7). For each class N_i choose β_i as the lightest item in $\{1,\ldots,\ell_i\}$ and delete items $j \neq \beta_i$ with $w_{ij} \geq w_{i\beta_i}$; 2) If $W'' < c$ then the slope $\lambda = (\delta\bar{p}/\delta\bar{w})$ was too large. For each class N_i choose α_i as the heaviest item in $\{1,\ldots,\ell_1\}$ and delete items $j \neq \alpha_i$ with $p_{ij} \leq p_{i\alpha_i}$ (items j with $w_{ij} \leq w_{i\alpha_i}$ are too light, and items with $w_{ij} > w_{i\alpha_i}$, $p_{ij} \leq p_{i\alpha_i}$ are dominated). If the class N_i has only one item left, save the LP-optimal choice $b_i = \beta_i$ and set $W = W + w_{ib_i}$, $P = P + p_{ib_i}$, then delete class N_i. Goto Step 1.

Depending on the choice of M in Step 1, we obtain different behavior of the algorithm. The best performance is obtained by choosing λ as the median of all slopes λ_i, $i = 1,\ldots,k$ (i.e. choose $M = k$) but for practical purpose $M \leq 15$ works well. Note that in the KP case, Algorithm 2 becomes the partitioning algorithm of Balas and Zemel (1980).

Proposition 3. *If we choose* $\lambda = (\delta\bar{p}/\delta\bar{w})$ *as the exact median of* M *different slopes* $\lambda_i = (\delta p_i/\delta w_i)$ *in Step* 1 *of Algorithm* 2, *at least* $\lceil M/2 \rceil$ *items are deleted at each iteration.*

Proof. Since λ is the median of the M classes, we have $\lambda_i \leq \lambda$ for $\lceil M/2 \rceil$ classes, so for these classes at least one item $j \neq \alpha_i$ exists which maximizes (13). Similarly we have $\lambda_i \geq \lambda$ for $\lceil M/2 \rceil$ classes, so for these classes at least one item $j \neq \beta_i$ exists which maximizes (13). If $W' > c$ in Step 5, at least $\lceil M/2 \rceil$ items $\{\beta_i\}$ will be deleted. Otherwise if $W'' < c$, at least $\lceil M/2 \rceil$ items $\{\alpha_i\}$ will be deleted. □

Corollary 1. *If* $M = 1$, *at least one item is deleted at each iteration of Algorithm* 2, *yielding a complexity of* $O(n^2)$.

Corollary 2. *If* $M = k$ *and the size of each class* n_i *is bounded by a constant* K, *Algorithm* 2 *runs in* $O(n)$.

Proof. Due to Proposition 3 at least $\lceil k/2 \rceil$ items are deleted at each iteration. Since n_i is bounded by K it means that at least $\lceil n/(2K) \rceil$ items are deleted at each iteration, yielding the complexity. □

4. Expanding core

Considering the KP, Balas and Zemel (1980) proposed to enumerate only a small amount of the items – the so-called *core* – where there was a large probability for finding an optimal solution. However the

core cannot be identified a priori, implying that in some cases optimality of the core solution cannot be proved, and thus a complete enumeration has to be performed.

Plateau and Elkihel (1985) noted, that even though the core cannot be identified *before* KP is solved, it can be identified *while* the problem is solved by using an *expanding core*. This result was improved by Pisinger (1995c) who showed that a minimal core may be obtained by using dynamic programming, as the breadth-first search implies that all variations of the solution vector have been tested before a new variable is added to the core.

We will use the same concept for MCKP, but now the core consists of the smallest possible number of *classes* N_i, such that an optimal solution may be determined and proved. Where the core for KP naturally consists of items having profit-to-weight ratio close to that of the break item, there is no natural way of ordering the classes in MCKP. Instead we use the *gradients* to identify a core: Define the positive and negative gradient λ_i^+ and λ_i^- for each class N_i, $i \neq a$, by (7) and (8). Due to (12) we have that

$$\lambda_i^+ \leq (\delta\bar{p}/\delta\bar{w}) \leq \lambda_i^-. \tag{14}$$

Order the sets $L^+ = \{\lambda_i^+\}$ according to nonincreasing values, and $L^- = \{\lambda_i^-\}$ according to nondecreasing values. Initially the core C only consists of the fractional class N_a, and then we repeatedly add classes N_i corresponding to the next gradient from the ordered sets L^+ and L^-. Since each class occur twice (once in each set L^+ and L^-), we pass over a class if it already has been considered.

4.1. Class reduction

Before adding a class N_i to the core C it is appropriate to fathom unpromising items from the class. We check whether each item $j \in N_i$ has an upper bound larger than the currently best solution z. For this purpose we use an upper bound obtained by relaxing the constraint on the fractional variables $b_a, b'_a \in N_a$ from $x_{b_a}, x_{b'_a} \in \{0, 1\}$ to $x_{b_a}, x_{b'_a} \in \mathbb{R}$ in (1). The upper bound on item $j \in N_i$ is then

$$u_{ij} = P - p_{ib_i} + p_{ij} + \lambda(c - W + w_{ib_i} - w_{ij}), \tag{15}$$

and if $u_{ij} < z + 1$ we may fix x_{ij} to 0. Since the bound (15) is evaluated in constant time, the complexity of reducing class N_i is $O(n_i)$.

If the reduced set N'_i has only one item left, we fathom the class, since no choices have to be done. Otherwise we order the items in N'_i according to nondecreasing weights and delete dominated items by applying (3). The computational effort is concentrated on the sorting, yielding a complexity of $O(n'_i \log n'_i)$ where n'_i is the size of N'_i. In Section 8 it will be demonstrated that a large majority of the items may be fixed at their optimal value by the reduction (15), thus significantly decreasing the number of items which need to be sorted.

5. A dynamic programming algorithm

The *core* is a set of currently enumerated classes $C = \{N_{r_1}, \ldots, N_{r_m}\}$. We will use dynamic programming for this enumeration, thus let $f_C(\tilde{c})$, $\tilde{c} = 0, \ldots, 2c$, be an optimal solution to the following core problem, where variables in classes outside the core are fixed at their LP-optimal values:

$$f_C(\tilde{c}) = \max\left\{ \begin{array}{l} \sum_{N_i \in C} \sum_{j \in N_i} p_{ij}x_{ij} + \sum_{N_i \notin C} p_{ib_i}: \\ \sum_{N_i \in C} \sum_{j \in N_i} w_{ij}x_{ij} + \sum_{N_i \notin C} w_{ib_i} \leq \tilde{c}, \\ \sum_{j \in N_i} x_{ij} = 1 \text{ for } N_i \in C,\ x_{ij} \in \{0, 1\} \end{array} \right\}. \tag{16}$$

For an empty core $C = \emptyset$ we set $f_\emptyset(\tilde{c}) = \sum_{i=1}^{k} p_{ib_i}$ for all $\tilde{c} \geq \sum_{i=1}^{k} w_{ib_i}$, and $f_\emptyset(\tilde{c}) = -\infty$ for all smaller values of $\tilde{c}$. Each time the core is extended with a new class N_i, then $f_{C+N_i}(\tilde{c})$ can be found by the recursion

$$f_{C+N_i}(\tilde{c}) = \max \begin{cases} f_C(\tilde{c} - w_{i1} + w_{ib_i}) + p_{i1} - p_{ib_i} & \text{if } 0 \leq \tilde{c} - w_{i1} + w_{ib_i} \leq 2c, \\ f_C(\tilde{c} - w_{i2} + w_{ib_i}) + p_{i2} - p_{ib_i} & \text{if } 0 \leq \tilde{c} - w_{i2} + w_{ib_i} \leq 2c, \\ \vdots & \\ f_C(\tilde{c} - w_{in_i} + w_{ib_i}) + p_{in_i} - p_{ib_i} & \text{if } 0 \leq \tilde{c} - w_{in_i} + w_{ib_i} \leq 2c. \end{cases} \tag{17}$$

An optimal solution to MCKP is found as $z = f_C(c)$ for a complete core $C = \{N_1, \ldots, N_k\}$, and we obtain $z = -\infty$ if assumption (2) is violated. Since the variables not in the core are fixed at their LP-optimal values, we must accept capacities $\tilde{c} > c$ in a transition stage, as these states may become feasible at a later stage. However let

$$V = \sum_{N_i \notin C} \left(w_{ib_i} - \min_{j \in N_i} w_{ij} \right) \tag{18}$$

be the largest weight sum that can be released in classes outside the core. Thus only states with $\tilde{c} \leq c + V \leq 2c$ need to be considered in the recursion.

The recursion (17) demands $O(n_i)$ operations for each class in the core and for each capacity $\tilde{c}$, yielding the complexity $O(\sum_{i=1}^{k} 2cn_i) = O(nc)$ for a complete enumeration. However if optimality of a state can be proved, we may terminate the enumeration instantly. In this case the computational effort is $O(c \sum_{N_i \in C} n_i)$, which is very efficient for small core sizes. The ordering of the classes according to gradients ensures that generally only a few dozen of classes need to be enumerated even for very large instances.

The traditional recursion for MCKP as presented in Martello and Toth (1990) reaches an optimal solution by only considering feasible capacities $\tilde{c} \leq c$ as the classes are enumerated. This approach has the drawback that a solution is not reached before all classes have been enumerated, meaning that we have to pass through all $O(nc)$ steps.

The space complexity of recursion (17) is $O(kc)$, as for each class we only need to save the index of the chosen item. Thus for a given core C let the set of partial vectors be given by

$$Y_C = \{(y_1, \ldots, y_m) : y_i \in \{1, \ldots, n_{r_i}\},\ i = 1, \ldots, m\}, \tag{19}$$

where each variable y_i determines that variable $x_{iy_i} = 1$ while the remaining binary variables in N_i are set to zero. The weight and profit sum of a vector $\bar{y}_i = (y_1, \ldots, y_m) \in Y_C$ corresponds to the weight and profit sum of the chosen variables y_{r_i} when $N_{r_i} \in C$, and to the LP-optimal choices b_i when $N_{r_i} \notin C$. Thus

$$\mu_i = \sum_{N_i \in C} w_{iy_i} + \sum_{N_i \notin C} w_{ib_i}, \tag{20}$$

$$\pi_i = \sum_{N_i \in C} p_{iy_i} + \sum_{N_i \notin C} p_{ib_i}. \tag{21}$$

It is convenient to represent each vector $\bar{y}_i \in Y_C$ by a *state* (μ_i, π_i, v_i), where μ_i, π_i are given above, and v_i is a (not necessarily complete) representation of $\bar{y}_i$. As only undominated states are considered we have $f_C(\mu_i) = \pi_i$. An iterative version of recursion (17) is presented in Pisinger (1994).

5.1. Reduction of states

Although the number of states in Y_C at any time is bounded by $2c$, the enumeration may be considerably improved by applying some upper bound tests in order to delete unpromising states.

Assume that the core C is obtained by adding classes corresponding to the first m gradients from L^- and L^+ and that N_s and N_t are the next classes to be added from each set. Thus the gradients satisfy

$$\max_{N_i \notin C} \lambda_i^- \leq \lambda_s^-, \tag{22}$$

$$\min_{N_i \notin C} \lambda_i^+ \geq \lambda_t^+. \tag{23}$$

By this assumption we get the following upper bound on a state i given by (μ_i, π_i, v_i):

$$u(i) = \begin{cases} \pi_i + (c - \mu_i)\lambda_t^+ & \text{if } \mu_i \leq c, \\ \pi_i + (c - \mu_i)\lambda_s^- & \text{if } \mu_i > c. \end{cases} \tag{24}$$

For conveniency we set $\lambda_t^+ = 0$ if the set L^+ is empty, and $\lambda_s^- = \infty$ if L^- is empty, ensuring that states which cannot be improved further are fathomed. Note that this bound is derived in constant time, where the linear upper bound presented by Zemel (1980) demands $O(n)$ time, improved to $O(k \log^2(n/k))$ by Dudzinski and Walukiewicz (1984).

The bound (24) may also be used for deriving a global upper bound on MCKP. Since any optimal solution must follow a branch in Y_C, the global upper bound corresponds to the upper bound of the most promising branch in Y_C. Therefore a global upper bound on MCKP is given by

$$u_{\text{MCKP}} = \max_{\bar{y}_i \in Y_C} u(i). \tag{25}$$

Since the gradient λ_t^+ will be decreasing during the solution process, and the gradient λ_s^- will be increasing, u_{MCKP} will become more and more tight as the core is expanded. For a complete core $C = \{N_1, \ldots, N_k\}$ we get $u_{\text{MCKP}} = z$ for the optimal solution z. This observation may be used for deriving an approximate algorithm for MCKP, as the enumeration simply is halted when the current lower bound is sufficiently close to u_{MCKP}.

6. Finding the solution vector

According to the principles of dynamic programming, the optimal solution vector x^* should be found by backtracking through the sets of states, implying that all sets of states should be saved during the solution process. In the computational experiments it is demonstrated that the number of states may be half a million in each iteration and since the number of classes may be large ($k = 10\,000$) we would need to store billions of states. Pisinger (1995c) proposed to save only the last A changes in the solution vector in each state (μ, π, v). If this information is not sufficient for reconstructing the solution vector, we simply solve a new MCKP problem with a reduced number of variables. This is repeated till the solution vector is completely defined. More precisely we do the following:

Assume that v consists of A pairs (i, j), indicating that the variable x_{ij} was chosen in class N_i. Whenever an improved solution is found during the enumeration of Y_C, we save the corresponding state (μ, π, v). When the algorithm terminates, all variables are set to the break solution $x_{ib_i} = 1$ for $i = 1, \ldots, k$ and $x_{ij} = 0$ for $i = 1, \ldots, k$, $j \neq b_i$. Then we make the changes registered in v:

$$x_{ij} = 1,\ x_{ib_i} = 0 \quad \text{for } (i, j) \in v,$$

$$\mu' = \mu + \sum_{(i,j) \in v} (w_{ib_i} - w_{ij}), \qquad \pi' = \pi + \sum_{(i,j) \in v} (p_{ib_i} - p_{ij}). \tag{26}$$

If the backtracked weight and profit sums μ', π' correspond to the weight and profit sums W, P of the break solution, we know that the obtained vector is correct. Otherwise we solve a new MCKP, this time with capacity $c = \mu'$, lower bound $z = \pi' - 1$, and global upper bound $u = \pi'$. The process is repeated until the solution vector x is completely defined. The technique has proved very efficient, since generally only a few iterations are needed. With $A = 10$, a maximum of 4 iterations has been observed for large data instances, but usually the optimal solution vector is found after the first iteration.

7. Main algorithm

The previous sections may be summed up to the following main algorithm:

Algorithm 3
procedure mcknap;
Solve LMCKP *through a partitioning algorithm.*
Determine gradients $L^+ = \{\lambda_i^+\}$ *and* $L^- = \{\lambda_i^-\}$ *for* $i = 1, \ldots, k$, $i \neq a$.
Partially sort L^+ *in decreasing order and* L^- *in increasing order.*
$z := 0$; $s := 1$; $t := 1$; $C := \{N_a\}$; $Y_C :=$ reduceclass(N_a);
repeat
  reduceset(Y_C); **if** ($Y_C = \emptyset$) **then break**; **fi**;
  $N_i := L_s^-$; $s := s + 1$; {*Choose next class from* L^-}
  if (N_i *is not used*) **then**
    $R_i :=$ reduceclass(N_i);
    if ($|R_i| > 1$) **then** add(Y_C, R_i);
  fi;
  reduceset(Y_C); **if** ($Y_C = \emptyset$) **then break**; **fi**;
  $N_i := L_t^+$; $t := t + 1$; {*Choose next class from* L^+}
  if (N_i *is not used*) **then**
    $R_i :=$ reduceclass(N_i);
    if ($|R_i| > 1$) **then** add(Y_C, R_i);
  fi;
forever;
Find the solution vector.

The first step of the algorithm is to solve the LMCKP as sketched in Section 3. Hereby we obtain the fractional class N_a, the break solution $\{b_i\}$ as well as the corresponding weight and profit sum W and P.

The gradients λ_i^+ and λ_i^- are determined and the sets L^+ and L^- are ordered. Since we initially do not need a complete ordering, we use a partial ordering as presented in Pisinger (1995a): Using the quicksort algorithm for sorting (Hoare, 1962), we always choose the interval containing largest values (resp. smallest for L^-) for further partitioning, while the other interval is pushed onto a stack. In this way we continue until the largest (resp. smallest) values have been determined. Later in Algorithm 3, if more values are needed, we simply pop the next interval from the stack by need and partition it further. Thus for small core sizes we use linear time for this ordering.

Our initial core is the fractional class N_a, which is reduced by procedure 'reduceclass'. Here we apply criterion (15) to fix as many variables as possible at their optimal value. If the reduced class has more than one item left, we sort the items according to increasing weight, and then apply criterion (3) to remove dominated items. Hereby we obtain the reduced class R_a which is the current set of states Y_C.

The set of states Y_C is reduced by procedure 'reduceset' which apply criterion (24) to fathom unpromising states. Moreover the procedure checks whether any feasible state ($\mu \leq c$) has improved the lower bound z, and updates the current best solution in that case.

Now we alternately include classes from L^+ and L^-, each time reducing the class to see if it must be added to the core. The reduced class R_i is added to the set of states Y_C by using recursion (17), indicated by procedure 'add' above.

The iteration stops when no more states are feasible, meaning that no improvements can occur. Note that we set $\lambda_t^+ = 0$ when L^+ is empty, and $\lambda_s^- = \infty$ when L^- is empty, meaning that the iteration in any case will stop when all classes have been considered.

7.1. Minimality

We will show that Algorithm 3 solves MCKP to proven optimality with a minimal core and with minimal effort for sorting and reduction. More precisely we have:

Definition 2. Given a core C and the corresponding set of states Y_C. We say that the core problem has been solved to *proven optimality* if one (or both) of the following cases occur: 1) $z = u_{\text{MCKP}}$ where z is the best feasible solution in Y_C; 2) All classes $N_i \notin C$ could be reduced to contain only the LP-optimal choice b_i.

Note that if $z = u_{\text{MCKP}}$ then all states in Y_C will be fathomed by (24), implying that $Y_C = \emptyset$. Thus the definition states, that we cannot prove optimality before the enumeration terminates or all variables outside the core can be fixed at their LP-optimal values.

Since MCKP is NP-hard we may assume that the enumeration cannot be guided by other principles than *greed*. Thus if a problem can be solved to optimality with final gradients λ_s^- and λ_t^+ then we may assume that all classes with smaller λ_i^- and larger λ_i^+ also have been considered, as they have a smaller loss per weight unit when making changes from the LP-optimal value. This leads to the following

Definition 3. MCKP has been solved with a *minimal core* if the following invariant holds: A class N_s (resp. N_t) is only added to the core C if the corresponding core problem could not be solved to proven optimality, and the set N_s (resp. N_t) has the smallest gradient λ_s^- (resp. largest gradient λ_t^+).

The definition ensures that if MCKP has been solved to optimality with a minimal core C, no *subset* core $C' \subset C$ exists, such that a fixed-core algorithm can solve the problem to optimality. Anyway a smaller *sized* core C' may exist if $C \not\subset C'$ and $C' \not\subset C$, but according to our definition such cores are not comparable.

Definition 4. The *sorting* effort has been minimal if 1) A class N_i is sorted only when the current core C could not be solved to optimality; 2) N_i is the next class to be included according to Definition 3; and 3) only items which have passed the reduction criterion (24) are sorted.

Definition 5. The effort used for *reduction* has been minimal if a class N_i is reduced only when the core C could not be solved to optimality, and N_i is the next class to be included according to the rule in Definition 3.

Proposition 4. *The presented algorithm solves* MCKP *with a minimal core, using minimal sorting and reduction effort (with the mentioned order of priority).*

Proof. In each iteration of Algorithm 3 we test whether the current core problem has been solved to proven optimality: The breadth-first search ensures that all variations of Y_C have been tested, and thus z is the best feasible solution in C. If $Y_C = \emptyset$ we terminate the algorithm, while a new class is only added to the core if some variables could not be fixed at their LP-optimal values. Thus at any step, no subset core exists such that the problem can be solved to proven optimality. As we follow the greedy principle for adding classes to the core, this proves the minimality of the core.

The sorting and reduction is done by need in Algorithm 3, exactly as described in Definitions 4 and 5. □

For a more general discussion of minimal knapsack algorithms, see Pisinger (1995b).

8. Computational experiments

The presented algorithm has been implemented in C, and a complete listing is available from the author on request. The following results have been achieved on a HP9000/730 computer.

We will consider how the algorithm behaves for different problem sizes, test instances, and data-ranges. Five types of randomly generated data instances are considered, each instance tested with *data-range* $R_1 = 1\,000$ or $R_2 = 10\,000$ for different number of classes k and sizes n_i:

- *Uncorrelated data instances* (UC): In each class we generate n_i items by choosing w_{ij} and p_{ij} randomly in $[1, R]$.
- *Weakly correlated data instances* (WC): In each class, w_{ij} is randomly distributed in $[1, R]$ and p_{ij} is randomly distributed in $[w_{ij} - 10, w_{ij} + 10]$, such that $p_{ij} \geq 1$.
- *Strongly correlated data instances* (SC): For KP these instances are generated as w_j randomly distributed in $[1, R]$ and $p_j = w_j + 10$, which are very hard indeed. Such instances are trivial for MCKP, since they degenerate to subset-sum data instances, but hard instances for MCKP may be constructed by cumulating strongly correlated KP-instances: For each class generate n_i items (w'_j, p'_j) as for KP, and order these by increasing weight. The data instance for MCKP is then $w_{ij} = \sum_{h=1}^{j} w'_h$, $p_{ij} = \sum_{h=1}^{j} p'_h$, $j = 1, \ldots, n_i$. Such instances have no dominated items, and form an upper convex set.
- *Subset-sum data instances* (SS): w_{ij} randomly distributed in $[1, R]$ and $p_{ij} = w_{ij}$. Such instances are hard since any upper bound will yield $u_{ij} = c$.
- *Sinha and Zoltners* (SZ): Sinha and Zoltners (1979) constructed their instances in a special way. For each class construct n_i items as (w'_j, p'_j) randomly distributed in $[1, R]$. Order the profits and weights in increasing order, and set $w_{ij} = w'_j$, $p_{ij} = p'_j$, $j = 1, \ldots, n_i$. Note that such data instances have no dominated items.

The constant M in Algorithm 2 is chosen as $M = 15$ and for each data instance the capacity c is

$$c = \tfrac{1}{2} \sum_{i=1}^{k} \left(\min_{j \in N_i} w_{ij} + \max_{j \in N_i} w_{ij} \right). \tag{27}$$

We construct and solve 100 different data instances for each problem type, size and range. The presented results are average values or extreme values.

First Table 1 shows the average core size (measured in classes) for solving MCKP to optimality. For most instances only a few classes need to be considered in the dynamic programming. The strongly correlated data instances however demand that almost all classes are considered. Table 2 shows how many classes have been tested by criterion (15). It is seen, that when many classes are present, only a few percent of the classes are reduced, meaning that we may solve the problem to optimality without even

Table 1
Final core-size. Average of 100 instances

k	n_i	UC		WC		SC		SS		SZ	
		R_1	R_2	R_1	R_2	R_1	R_2	R_1	R_2	R_1	R_2
10	10	2	2	8	8	8	9	2	4	6	5
100	10	8	9	11	16	85	84	2	4	17	17
1000	10	15	20	7	12	791	775	0	2	18	33
10000	10	10	28	1	10	7563	7800	0	0	11	33
10	100	2	3	4	5	8	8	1	2	7	8
100	100	7	10	3	6	84	95	0	1	15	34
1000	100	6	17	1	4	839	915	0	0	11	41
10	1000	1	2	2	2	4	8	0	1	6	9
100	1000	1	6	0	2	25	82	0	0	9	30

Table 2
Percentage of all classes which have been tested by weak upper bound. Average of 100 instances

k	n_i	UC		WC		SC		SS		SZ	
		R_1	R_2	R_1	R_2	R_1	R_2	R_1	R_2	R_1	R_2
10	10	52	55	87	88	85	88	23	37	83	82
100	10	46	63	14	19	87	86	2	4	68	80
1000	10	20	52	1	1	82	82	0	0	18	70
10000	10	0	26	0	0	78	80	0	0	0	19
10	100	42	60	43	55	81	81	10	19	80	90
100	100	23	56	3	6	84	95	0	1	23	82
1000	100	1	29	0	0	84	92	0	0	2	21
10	1000	10	48	16	20	45	79	0	11	66	97
100	1000	1	20	0	2	25	82	0	0	11	39

considering a large majority of the classes. The strongly correlated data instances again demonstrate that almost all classes must be considered.

The efficiency of the weak upper bound (15) is given in Table 3. The entries show how many percent of the tested items which are reduced. Generally a large majority of the variables are fixed to their

Table 3
Percentage of tested items which are reduced. Average of 100 instances

k	n_i	UC		WC		SC		SS		SZ	
		R_1	R_2	R_1	R_2	R_1	R_2	R_1	R_2	R_1	R_2
10	10	83	84	48	27	45	34	0	0	70	73
100	10	88	88	62	56	51	51	0	0	86	86
1000	10	89	90	68	49	53	54	0	0	88	89
10000	10	86	90	80	68	50	52	0	0	72	90
10	100	98	98	75	61	84	79	0	0	86	85
100	100	99	99	87	68	85	85	0	0	93	97
1000	100	98	99	94	86	84	85	0	0	94	98
10	1000	100	100	87	58	50	94	0	0	89	90
100	1000	100	100	94	85	50	94	0	0	93	96

Table 4
Largest set of states Y_C in dynamic programming, measured in thousands. Maximum of 100 instances

k	n_i	UC		WC		SC		SS		SZ	
		R_1	R_2	R_1	R_2	R_1	R_2	R_1	R_2	R_1	R_2
10	10	0	0	1	10	3	24	4	47	0	0
100	10	0	0	4	52	7	68	4	38	0	0
1000	10	1	0	4	39	20	194	0	28	2	3
10000	10	1	4	5	46	84	572	0	0	4	12
10	100	0	0	4	40	1	10	3	28	1	1
100	100	0	0	4	40	4	26	3	28	3	3
1000	100	0	1	3	43	10	106	0	0	4	8
10	1000	0	0	3	35	3	4	0	30	3	10
100	1000	0	0	3	36	25	9	0	20	4	31

Table 5
Total computing time in seconds. Average of 100 instances

k	n_i	UC		WC		SC		SS		SZ	
		R_1	R_2	R_1	R_2	R_1	R_2	R_1	R_2	R_1	R_2
10	10	0.00	0.00	0.01	0.05	0.01	0.09	0.01	0.17	0.00	0.00
100	10	0.00	0.00	0.02	0.28	0.37	5.16	0.01	0.11	0.01	0.01
1000	10	0.03	0.03	0.03	0.23	7.30	92.46	0.01	0.09	0.04	0.05
10000	10	0.25	0.31	0.24	0.42	169.94	1628.57	0.17	0.17	0.33	0.41
10	100	0.00	0.00	0.03	0.58	0.02	0.19	0.06	1.05	0.01	0.02
100	100	0.02	0.02	0.03	0.55	0.33	6.93	0.01	0.68	0.05	0.07
1000	100	0.14	0.17	0.16	0.43	9.57	195.75	0.13	0.13	0.24	0.32
10	1000	0.02	0.03	0.12	2.75	1.64	0.14	0.02	12.55	0.19	0.74
100	1000	0.12	0.15	0.18	1.11	173.69	2.97	0.13	0.15	0.41	2.66

optimal value this way. To illustrate the hardness of the dynamic programming, we measure the largest size of Y_C for each data instance in Table 4. It is seen that strongly correlated data instances may result in more than half a million states. Still this is far less than the space bound $O(2c)$.

Finally Table 5 gives the average computational times. Easy data instances are solved in a fraction of a second. Only the strongly correlated instances demand more computational effort, but are still solved within 30 minutes. For comparison it should be mentioned that Sinha and Zoltners (1979) solve SZ type problems of size $k = 50$, $n_i = 10$ in 0.12 seconds, while Armstrong et. al. (1983) solve the same problems of size $k = 400$, $n_i = 100$ in 2.71 seconds. Both references generate the weights in a small range $R < 100$ meaning that very little enumeration is necessary to obtain an optimal solution.

The above results indicate that the presented algorithm outperforms any algorithm for MCKP, implying that the stated minimal properties actually cause drastical reductions in the computational times. More computational experiments with the presented algorithm can be found in Pisinger (1994).

9. Conclusions

We have presented a complete algorithm for the exact solution of the Multiple-Choice Knapsack Problem. To our knowledge, it is the first enumerative algorithm which makes use of the partitioning algorithms by Dyer (1984) and Zemel (1984). In order to do this, it has been necessary to derive new

upper bounds based on the positive and negative gradients, as well as choosing a strategy for which classes should be added to the core.

The algorithm satisfies some minimality constraints as defined in Section 7.1: It solves MCKP with a minimal core, since variables only are added to the core if the current core could not be solved to optimality, and the effort used for sorting and reduction is also minimal according to the stated definitions.

The computational complexity is $O(n + c\Sigma_{N_i \in C} n_i)$ for a minimal core C, thus we have a linear solution time for small cores, and pseudopolynomial solution time for large cores. Computational experiments document that the presented algorithm is indeed very efficient. Even very large data instances are solved in a fraction of a second; only strongly correlated data instances demand more computational effort.

References

Armstrong, R.D., Kung, D.S., Sinha, P., and Zoltners, A.A. (1983), "A computational study of a multiple-choice knapsack algorithm", *ACM Transactions on Mathematical Software* 9, 184–198.

Balas, E., and Zemel, E. (1980), "An algorithm for large zero-one Knapsack Problems", *Operations Research* 28, 1130–1154.

Dantzig, G.B. (1957), "Discrete variable extremum problems", *Operations Research* 5, 266–277.

Dudzinski, K., and Walukiewicz, S. (1984), "A fast algorithm for the linear multiple-choice knapsack problem", *Operations Research Letters* 3, 205–209.

Dudzinski, K., and Walukiewicz, S. (1987), "Exact methods for the knapsack problem and its generalizations", *European Journal of Operational Research* 28, 3–21.

Dyer, M.E. (1984), "An O(n) algorithm for the multiple-choice knapsack linear program", *Mathematical Programming* 29, 57–63.

Dyer, M.E., Kayal, N., and Walker, J. (1984), "A branch and bound algorithm for solving the multiple choice knapsack problem", *Journal of Computational and Applied Mathematics* 11, 231–249.

Fayard, D., and Plateau, G. (1977), "Reduction algorithm for single and multiple constraints 0–1 linear programming problems", Conference on Methods of Mathematical Programming, Zakopane, Poland.

Fisher, M.L. (1981), "The Lagrangian relaxation method for solving integer programming problems", *Management Science* 27, 1–18.

Hoare, C.A.R. (1962), "Quicksort", *Computer Journal* 5, 1, 10–15.

Martello, S., and Toth, P. (1988), "A new algorithm for the 0–1 Knapsack Problem", *Management Science* 34, 633–644.

Martello, S., and Toth, P. (1990), *Knapsack Problems: Algorithms and Computer Implementations*, Wiley, Chichester, UK.

Nauss, R.M. (1978), "The 0–1 knapsack problem with multiple choice constraint", *European Journal of Operational Research* 2, 125–131.

Pisinger, D. (1994), "A minimal algorithm for the multiple-choice knapsack problem", Report 94/25 DIKU, University of Copenhagen, Denmark.

Pisinger, D. (1995a), "An expanding-core algorithm for the exact 0–1 knapsack problem", to appear in *European Journal of Operational Research*.

Pisinger, D. (1995b), "A minimal algorithm for the bounded knapsack problem", in: E. Balas and J. Clausen (eds.), *Proceedings IPCO IV*, Lecture Notes in Computer Science, Springer, Berlin.

Pisinger, D. (1995c), "A minimal algorithm for the 0–1 knapsack problem", submitted to *Operations Research*, under revision.

Plateau, G., and Elkihel, M. (1985), "A hybrid method for the 0–1 knapsack problem", *Methods of Operations Research* 49, 277–293.

Sinha, A., and Zoltners, A.A. (1979), "The multiple-choice knapsack problem", *Operations Research* 27, 503–515.

Witzgal, C. (1977), "On one-row linear programs", Applied Mathematics Division, National Bureau of Standards.

Zemel, E. (1980), "The linear multiple choice knapsack problem", *Operations Research* 28, 1412–1423.

Zemel, E. (1984), "An O(n) algorithm for the linear multiple choice knapsack problem and related problems", *Information Processing Letters* 18, 123–128.