Operations Research Letters 54 (2024) 107104

Contents lists available at ScienceDirect

Operations Research Letters

journal homepage: www.elsevier.com/locate/orl

On upper bounds for the multiple knapsack assignment problem

Laura Galli [a], Adam N. Letchford [b,*]

[a] Department of Computer Science, University of Pisa, Largo B. Pontecorvo 3, 56127 Pisa, Italy
[b] Department of Management Science, Lancaster University, Lancaster LA1 4YW, United Kingdom

ARTICLE INFO

Article history:
Received 14 July 2023
Received in revised form 27 December 2023
Accepted 28 February 2024
Available online 6 March 2024

Keywords:
Combinatorial optimisation
Knapsack problems
Multiple assignment knapsack problem

ABSTRACT

The *Multiple Knapsack Assignment Problem* is a strongly $\mathcal{NP}$-hard combinatorial optimisation problem, with several applications. We show that an *upper bound* for the problem, due to Kataoka and Yamada, can be computed in linear time. We then show that some bounds due to Martello and Monaci dominate the Kataoka-Yamada bound. Finally, we define an even stronger bound, which turns out to be particularly effective when the number of knapsacks is not a multiple of the number of item classes.

1. Introduction

Knapsack Problems form a fundamental family of combinatorial optimisation problems, and there is a huge literature on them (see [5,8] for detailed surveys). Here, we focus on the *Multiple Knapsack Assignment Problem* (MKAP), introduced by Kataoka and Yamada [4].

In the MKAP, we have a set M of *knapsacks* and a set N of *items*. Each knapsack $i \in M$ has a *capacity* c_i. Each item $j \in N$ has a *profit* p_j and a *weight* w_j. Moreover, the set N is partitioned into r subsets or *classes*, which we call $N_1, \ldots, N_r$. Items from a given class can only be placed into a given knapsack if that knapsack has been "assigned" to that class. Each knapsack can be assigned to at most one class, but there is no restriction on the number of knapsacks assigned to any given class. The total weight of the items placed in any given knapsack must not exceed the capacity of that knapsack. The task is to assign each knapsack to a class, and place some items into each knapsack, in order to maximise the total profit.

The MKAP has applications in the purchasing of goods and their subsequent transportation by air, rail or water [3,4,7]. It is also strongly $\mathcal{NP}$-hard. Indeed, it reduces to the *multiple knapsack problem* (MKP) when $r = 1$, and the MKP was proven to be strongly $\mathcal{NP}$-hard in [11].

To our knowledge, there have only been four papers on the MKAP [3,4,6,7]. Here, we are primarily interested in Kataoka and Yamada [4] and Martello and Monaci [7], which present efficiently computable *upper bounds* for the MKAP.

We make four contributions. First, we show that the Kataoka-Yamada bound can be computed in linear time. Second, we show that the Martello-Monaci bounds dominate the Kataoka-Yamada bound. Third, we define an even stronger bound, and show that it can be computed in pseudo-polynomial time. Finally, we present the results of some computational experiments. Interestingly, it turns out that our bound is most useful when m is not a multiple of r.

The paper has a simple structure. Section 2 is a brief literature review. Section 3 contains the analysis of the existing bounds, and Section 4 presents the new bound. Section 5 gives the computational results, and Section 6 contains some concluding remarks.

Throughout the paper, we assume that the c_i and w_j are positive integers. We assume that $M = \{1, \ldots, m\}$ and $N = \{1, \ldots, n\}$, and we write $R = \{1, \ldots, r\}$. We assume without loss of generality that m and r are smaller than n. We let C denote $\sum_{i \in M} c_i$. For each $k \in R$, we let n_k, $P(k)$ and $W(k)$ denote $|N_k|$, $\sum_{j \in N_k} p_j$ and $\sum_{j \in N_k} w_j$, respectively. We also let $c_{\max}$ denote the maximum of $c_1, \ldots, c_m$ and $W_{\max}$ denote the maximum of $W(1), \ldots, W(r)$. Finally, *LP* and *DP* stand for "linear program" and "dynamic programming", respectively.

2. Literature review

We now give a brief review of the relevant literature. Subsection 2.1 recalls some results on the standard knapsack problem. Subsection 2.2 summarises the main papers on the MKAP, and Subsection 2.3 describes the existing upper-bounding procedures in detail.

* Corresponding author.
E-mail addresses: laura.galli@unipi.it (L. Galli), a.n.letchford@lancaster.ac.uk (A.N. Letchford).

https://doi.org/10.1016/j.orl.2024.107104

2.1. The knapsack problem

When $m = r = 1$, the MKAP reduces to the classical 0-1 knapsack problem (KP). The KP can be formulated as the following 0-1 LP:

$$\max \left\{ \sum_{j \in N} p_j x_j : \sum_{j \in N} w_j x_j \leq c,\ x \in \{0,1\}^n \right\}.$$

The KP is $\mathcal{NP}$-hard, but it can be solved in $O(nc)$ time using the DP algorithm of Bellman [2]. Moreover, the continuous relaxation of the 0-1 LP can be solved in $O(n)$ time [1].

2.2. The MKAP

Kataoka and Yamada [4] gave the following 0-1 LP formulation of the MKAP. For $i \in M$ and $j \in N$, the binary variable x_{ij} takes the value 1 if and only if item j is placed in knapsack i. For $i \in M$ and $k \in R$, the binary variable y_{ik} takes the value 1 if and only if knapsack i is assigned to class k. The 0-1 LP is then:

$$\max \quad \sum_{j \in N} p_j \sum_{i \in M} x_{ij} \tag{1}$$

$$\text{s.t.} \quad \sum_{j \in N_k} w_j x_{ij} \leq c_i y_{ik} \qquad (i \in M,\ k \in R) \tag{2}$$

$$\sum_{i \in M} x_{ij} \leq 1 \qquad (j \in N) \tag{3}$$

$$\sum_{k \in R} y_{ik} \leq 1 \qquad (i \in M) \tag{4}$$

$$x \in \{0,1\}^{mn},\ y \in \{0,1\}^{mr}. \tag{5}$$

The interpretation of the objective and constraints is straightforward.

Kataoka and Yamada provided a fast algorithm for solving the continuous relaxation of the 0-1 LP, along with a constructive heuristic. Lalla-Ruiz and Voß [6] presented a genetic algorithm for the MKAP. Dimitrov et al. [3] described an application of the MKAP in which all knapsacks have the same capacity. They designed an effective heuristic for this special case. Finally, Martello and Monaci [7] described an effective upper-bounding procedure based on surrogate relaxation. They also introduced a constructive heuristic and a metaheuristic.

2.3. Upper bounds for the MKAP

Let U_{KY} denote the upper bound that one obtains by solving the continuous relaxation of the 0-1 LP (1)-(5). Kataoka and Yamada provided a fast algorithm for computing U_{KY}. The idea is as follows. For each $k \in R$, define the following function:

$$z_k(\alpha) = \max \left\{ \sum_{j \in N_k} p_j x_j : \sum_{j \in N_k} w_j x_j \leq \alpha,\ x \in [0,1]^{n_k} \right\}, \tag{6}$$

where the domain of α is the closed interval $[0, C]$. It is easy to show that

$$U_{KY} = \max \left\{ \sum_{k \in R} z_k(u_k) : \sum_{k \in R} u_k \leq C,\ u \in [0,C]^r \right\}. \tag{7}$$

Kataoka and Yamada show that one can compute all of the functions z_k in $O(n \log n)$ time. They also show that (7) is a continuous optimisation problem with a separable, non-decreasing and concave objective function. Such problems can be solved efficiently, to any desired accuracy, with any of several available algorithms (see the survey [9]).

Martello and Monaci [7] point out that we can get a different upper bound by solving a "surrogate" KP of the form:

$$\max \left\{ \sum_{j \in N} p_j \tilde{x}_j : \sum_{j \in N} w_j \tilde{x}_j \leq C,\ \tilde{x} \in \{0,1\}^n \right\}. \tag{8}$$

This KP can be solved in $O(nC)$ time by DP. We will let U_{MM} denote the resulting upper bound.

Martello and Monaci also mention that one can sometimes decrease the knapsack capacities, without losing any feasible solutions. For $i \in M$ and $k \in R$, define

$$\bar{c}_{ik} = \max \left\{ \sum_{j \in S} w_j : \sum_{j \in S} w_j \leq c_i,\ S \subseteq N_k \right\}.$$

By definition, if knapsack i is assigned to class k, then the total weight of the items packed into knapsack i cannot exceed $\bar{c}_{ik}$. From this it follows that, for each $i \in M$, we can replace c_i with

$$\bar{c}_i = \max_{k \in R} \{\bar{c}_{ik}\}.$$

This in turn implies that, in the surrogate KP, we can replace C with $\bar{C} = \sum_{i \in M} \bar{c}_i$. The $\bar{c}_{ik}$ values can be computed in $O(nc_{\max})$ time via DP. We will let U_{MM}^{-} denote the resulting upper bound.

3. On the existing bounds

We begin by analysing the existing bounds. Subsections 3.1 and 3.2 concern the Kataoka-Yamada and Martello-Monaci bounds, respectively. Throughout this section, when we say "the LP relaxation", we mean the continuous relaxation of the 0-1 LP (1)-(5).

3.1. On the Kataoka-Yamada bound

Our first result concerns U_{KY}:

Theorem 1. *U_{KY} is equal to*

$$\max \left\{ \sum_{j \in N} p_j \tilde{x}_j : \sum_{j \in N} w_j \tilde{x}_j \leq C,\ \tilde{x} \in [0,1]^n \right\}. \tag{9}$$

Proof. First, we show that, given any feasible solution to the LP relaxation, we can construct a feasible solution to (9) with the same profit. So, let (x^*, y^*) be a solution to the LP relaxation. We construct a vector $\tilde{x}^* \in [0,1]^n$ by setting

$$\tilde{x}_j^* = \sum_{i \in M} x_{ij}^* \qquad (j \in N).$$

The fact that $\tilde{x}^*$ has the same profit as (x^*, y^*) is trivial. To see that $\tilde{x}^*$ is feasible for (9), note that:

$$\sum_{j \in N} w_j \tilde{x}_j^* = \sum_{i \in M} \sum_{j \in N} w_j x_{ij}^* \leq \sum_{i \in M} c_i \sum_{k \in R} y_{ik}^* \leq \sum_{i \in M} c_i = C,$$

where the first inequality comes from (2) and the second comes from (4).

We now show that, given any feasible solution to (9), we can construct a feasible solution to the LP relaxation with the same profit. Let $\tilde{x}^* \in [0,1]^n$ be a feasible solution to (9). We construct a pair (x^*, y^*) as follows. We define

$$u_k = \sum_{j \in N_k} w_j \tilde{x}_j^* \qquad (k \in R).$$

We then set x_{ij}^* to $c_i \tilde{x}_j^* / C$ for all i and j, and we set y_{ik}^* to u_k / C for all i and k. Note that, by construction, $\sum_{i \in M} x_{ij}^* = \tilde{x}_j^* \leq 1$ for

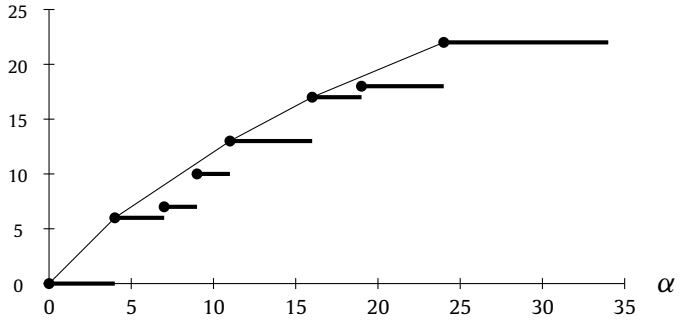

Fig. 1. Function z_k (narrow lines) and z_k^- (thick lines).

all $j \in N$. Thus, x^* satisfies (3), and it has the same profit as $\tilde{x}^*$. To see that (x^*, y^*) satisfies (2), note that the left-hand side of (2) evaluates to $u_k c_i / C$, and so does the right-hand side. Finally, to see that y^* satisfies (4), note that the sum of the u_k over all $k \in R$ cannot exceed C. □

The above theorem has the following useful corollary.

Corollary 1. $U_{MM} \leq U_{KY}$.

Proof. The problem (9) is a relaxation of the surrogate KP (8), obtained by replacing the binary condition $x \in \{0,1\}^n$ with the weaker condition $x \in [0,1]^n$. □

Thus, the Kataoka-Yamada bound is in general weaker than the Martello-Monaci bounds. The Kataoka-Yamada bound does however have one thing in its favour:

Corollary 2. *U_{KY} can be computed in $O(n)$ time, along with a vector $\tilde{x}^*$ that solves (9).*

Proof. As mentioned in Subsection 2.1, the continuous relaxation of a KP can be solved in $O(n)$ time. Moreover, computing C takes only $O(m)$ time, and we are assuming that $m \leq n$. □

We remark that the technique described in the proof of Theorem 1 enables one to convert the vector $\tilde{x}^*$ into an optimal pair (x^*, y^*) if desired. In this way, one can obtain an explicit optimal solution of the LP relaxation in $O(nm)$ time.

3.2. On the Martello-Monaci bounds

For what follows, we will find it useful to have alternative characterisations of U_{MM} and U_{MM}^-. To this end, we define the following function for each class $k \in R$:

$$z_k^-(\alpha) = \max \left\{ \sum_{j \in N_k} p_j x_j : \sum_{j \in N_k} w_j x_j \leq \alpha,\ x \in \{0,1\}^{n_k} \right\}, \qquad (10)$$

where, as before, the domain of α is $[0, C]$. Comparing (10) with (6), we see that $z_k^-(\alpha) \leq z_k(\alpha)$ for all k and α. Moreover, by definition, the z_k^- are piecewise-constant and discontinuous in general. To make this clear, we give a small example.

Example: Suppose that $m = 3$ and $c = (10, 10, 14)$. Also suppose that, for some $k \in R$, we have $N_k = \{1, 2, 3, 4\}$, $w_1 = 4$, $w_2 = 5$, $w_3 = 7$, $w_4 = 8$, $p_1 = 6$, $p_2 = 4$, $p_3 = 7$ and $p_4 = 5$. We have $C = 34$, $W(k) = 24$ and $P(k) = 22$. The functions z_k and z_k^- are sketched in Fig. 1. □

We have the following result:

Proposition 1.

$$U_{MM} = \max \left\{ \sum_{k \in R} z_k^-(u_k) : \sum_{k \in R} u_k \leq C,\ u \in [0, C]^r \right\}. \qquad (11)$$

Proof. We begin by showing that, given an optimal solution to (11), we can find a feasible solution to (8) that has the same profit. So, let $u^* \in [0, C]^r$ be an optimal solution to (11). From the definition of the functions z_k^-, it follows that, for each $k \in R$, there exists a set of items $S_k \subseteq N_k$ whose total profit is $z_k^-(u_k^*)$ and whose total weight does not exceed u_k^*. We construct a vector $\tilde{x}^* \in \{0,1\}^n$ by setting $\tilde{x}_j^*$ to 1 if and only if $j \in S_k$ for some $k \in R$. The profit of this solution is

$$\sum_{k \in R} \sum_{j \in N_k} p_j \tilde{x}_j^* = \sum_{k \in R} \left(\sum_{j \in S_k} p_j \right) = \sum_{k \in R} z_k^-(u_k^*) = U_{MM}.$$

We also have:

$$\sum_{k \in R} \sum_{j \in N_k} w_j \tilde{x}_j^* = \sum_{k \in R} \left(\sum_{j \in S_k} w_j \right) \leq \sum_{k \in R} u_k^* \leq C.$$

Thus, the weight of $\tilde{x}^*$ does not exceed C, as required. The proof in the reverse direction is similar. □

In exactly the same way, one can show that

$$U_{MM}^- = \max \left\{ \sum_{k \in R} z_k^-(u_k) : \sum_{k \in R} u_k \leq \bar{C},\ u \in [0, C]^r \right\}. \qquad (12)$$

4. The new bound and how to compute it

In this section, we present a new upper bound for the MKAP. Subsection 4.1 defines the bound and shows that it dominates U_{MM}^-. Subsections 4.2 and 4.3 are concerned with how to compute the bound efficiently.

4.1. The new bound

Recall that the variable u_k represents the total amount of capacity allocated to class k. Recall also that we can reduce the knapsack capacities from c_i to $\bar{c}_i$, without losing any feasible MKAP solutions. From these two facts it follows that we can restrict each of the u_k variables to the following domain:

$$\Omega = \left\{ q \in \mathbb{Z}_+ : q = \sum_{i \in M'} \bar{c}_i \text{ for some } M' \subseteq M \right\}.$$

To make this definition clear, we use the same example as in Subsection 3.2.

Example (cont.): One can check that $\bar{c}_1 = \bar{c}_2 = 4 + 5 = 9$ and $\bar{c}_3 = 5 + 8 = 13$. Thus, $\bar{c} = (9, 9, 13)$ and $\Omega = \{0, 9, 13, 18, 22, 31\}$. Fig. 2 shows the function z_k^-, but with the domain $[0, C]$ replaced with the (much) smaller domain Ω. □

Our new upper bound for the MKAP, then, is as follows:

$$U_{GL} = \max \left\{ \sum_{k \in R} z_k^-(u_k) : \sum_{k \in R} u_k \leq \bar{C},\ u \in \Omega^r \right\}. \qquad (13)$$

We have the following result:

Theorem 2. *$U_{GL} \leq U_{MM}^-$, and this inequality can be strict.*

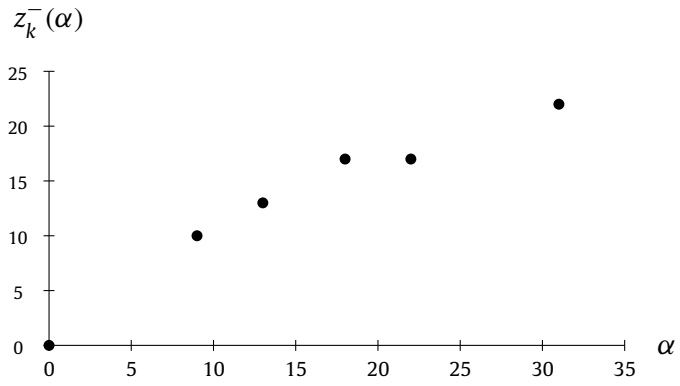

Fig. 2. Function z_k^- after restricting α to take values in Ω.

Proof. The fact that $U_{GL} \leq U_{MM}^-$ follows immediately from a comparison of (13) and (12). To show that the inequality can be strict, consider an MKAP instance with $m = 2$, $n = 6$, $r = 3$, $c_1 = c_2 = 3$, $N_1 = \{1, 2\}$, $N_2 = \{3, 4\}$, $N_3 = \{5, 6\}$, $p_1 = p_3 = p_5 = 10$, $p_2 = p_4 = p_6 = 1$, $w_1 = w_3 = w_5 = 2$ and $w_2 = w_4 = w_6 = 1$. One can check that there are three optimal solutions with a profit of $10 + 1 + 10 + 1 = 22$. The optimal solution to (11), on the other hand, has a profit of $10 + 10 + 10 = 30$. So $U_{MM} = 30$. Moreover, we have $\bar{c} = c$ and $\bar{C} = C$ for this instance, so $U_{MM}^- = 30$ as well. Now observe that $\Omega = \{0, 3, 6\}$. Moreover, we have $z_k^-(3) = z_k^-(6) = 10 + 1 = 11$ for $k = 1, 2, 3$. Thus, to obtain an optimal solution to (13), one must set two of the u_k variables to 3, and set the remaining u_k variable to 0. This gives $U_{GL} = 11 + 11 = 22$, which is optimal. □

4.2. Computing the new bound

We now show how to compute U_{GL} in pseudo-polynomial time. The first step is to compute the reduced knapsack capacities $\bar{c}_i$. As mentioned in Subsection 2.3, this can be done by DP in $O(nc_{\max})$ time. (With some care, one can implement the DP so that it takes only $O(c_{\max})$ space. We omit details for brevity.)

The next step is to use DP to compute and store the function z_k^- for all $k \in R$. Note that we only need to compute $z_k^-(\alpha)$ explicitly for $\alpha = 0, \ldots, W(k)$, since $z_k^-(\alpha) = P(k)$ for larger values of α. To do this for a given k, we use Algorithm 1. One can check that, for a given k, the algorithm runs in $O(n_k W(k))$ time and $O(W(k))$ space. Moreover, storing the functions themselves takes $O(rW_{\max})$ space.

Algorithm 1: Computing the z_k^- values for a given k.

input : class $k \in R$, profit p_j and weight w_j for each $j \in N_k$
Compute $W(k)$;
Create a 1-dimensional integer array Z of dimension $W(k) + 1$;
for $\alpha = 0, \ldots, W(k)$ **do**
  Set $Z[\alpha]$ to 0;
end
for $t = 1, \ldots, n_k$ **do**
  Let P and W be the profit and weight of the tth item in N_k;
  for $\alpha = W(k), W(k) - 1, \ldots, W$ **do**
    if $Z[\alpha - W] + P > Z[\alpha]$ **then**
      Set $Z[\alpha]$ to $Z[\alpha - W] + P$;
    end
  end
end
for $\alpha = 0, \ldots, W(k)$ **do**
  Set $z_k^-(\alpha)$ to $Z[\alpha]$;
end
output: $z_k^-(\alpha)$ for $\alpha = 0, \ldots, W(k)$

The next step is to use DP to compute the set Ω. To do this, we use Algorithm 2. One can check that $B(q)$ will be set to 'true' during the course of the algorithm if and only if $q \in \Omega$. One can also check that the algorithm runs in $O(m\bar{C})$ time and $O(\bar{C})$ space.

Algorithm 2: Computing the set Ω.

input : number of knapsacks m, reduced knapsack capacities $\bar{c}_1, \ldots, \bar{c}_m$
Compute $\bar{C}$;
Create a 1-dimensional Boolean array B of dimension $\bar{C} + 1$;
Set $B(0)$ to `true` and set Ω to $\emptyset$;
for $q = 1, \ldots, \bar{C}$ **do**
  Set $B(q)$ to `false`;
end
for $i = 1, \ldots, m$ **do**
  for $q = \bar{C}, \bar{C} - 1, \ldots, \bar{c}_i$ **do**
    if $B[q - \bar{c}_i] =$ `true` **then**
      Set $B[q]$ to `true`;
    end
  end
end
Let $\Omega = \{q : B[q] =$ `true`$\}$;
output: the set Ω

Finally, once we have computed the functions z_k^- and the set Ω, we have to solve (13) itself. To do this, we use DP yet again, as shown in Algorithm 3. In this algorithm, $P[t][q]$ represents the maximum total profit that can be obtained from the first t item classes, under the assumption that the total capacity allocated to those classes does not exceed q.

Algorithm 3: Solving the relaxation (13).

input : number of classes r, class weights $W(k)$,
total effective capacity $\bar{C}$, array containing the elements of Ω,
values $z_k^-(\alpha)$ for $k \in R$ and $\alpha = 0, \ldots, W(k)$
Create a 2-dimensional integer array P of dimension $r + 1$ by $\bar{C} + 1$;
for $t = 0, \ldots, r$ **do**
  for $q = 0, \ldots, \bar{C}$ **do**
    Set $P[t][q]$ to 0;
  end
end
for $t = 1, \ldots, r$ **do**
  for $q = 0, \ldots, \bar{C}$ **do**
    for *each* $s \in \Omega$ *such that* $q + s \leq \bar{C}$ **do**
      if $P[t-1][q] + z_t^-(s) > P[t][q+s]$ **then**
        Set $P[t][q+s]$ to $P[t-1][q] + z_t^-(s)$;
      end
    end
  end
end
Set U_{GL} to $P[r][\bar{C}]$;
output: Upper bound U_{GL}

One can check that Algorithm 3 runs in $O(r\bar{C}^2)$ time. Although it has a pseudo-polynomial running time, we have found that Algorithm 3 is in practice much slower than the algorithms mentioned above.

4.3. Speeding up the procedure

To speed up Algorithm 3, we need one additional piece of notation. For a given $k \in R$, we let L_k denote the *least element of* Ω *that is not less than* $W(k)$. So, for example, if $\Omega = \{0, 7, 9, 16\}$ and $W(k) = 8$, then $L_k = 9$. Note that, for any given k, the number of elements of Ω that do not exceed L_k is at most $W(k) + 1$.

We propose to compute the L_k values immediately after running Algorithm 2, as follows. We store the elements of Ω in an array, in increasing order of value. Then, for any given $k \in R$, we can compute L_k in $O(\log \bar{C})$ time, by binary search.

Now, by definition, $z_k^-(\alpha) = P(k)$ for all $\alpha \geq L_k$. Thus, when solving (13), we can assume that $u_k \leq L_k$ for all k. Thus, we can speed up Algorithm 3 by changing line 9 to "for each $s \in \Omega$ such that $q+s \leq \bar{C}$ and $s \leq L_t$". The effect of this change is that no more than $W(k)+1$ values of s need to be considered for any given t and q. From this it follows that the running time of Algorithm 3 decreases to $O(T\bar{C})$, where T is the total weight of the items. We have found this running time to be acceptable in practice.

5. Computational results

In this section, we give some computational results. Subsection 5.1 describes the test instances that we used, and Subsections 5.2 and 5.3 give results for some old and new instances, respectively.

We remark that all our procedures were coded in `C++` and compiled with `g++ 11.3.0`. The experiments were performed on a VM running on a 2.30 GHz Intel Xeon Gold 5218 CPU with 256Gb RAM, under `Ubuntu 22.4`. To solve the 0-1 KP (8) and its tightened version (with $\bar{C}$ in place of C), we simply used the mixed-integer optimizer of `CPLEX v.22.10`. (We could have solved the 0-1 KPs faster using a dedicated algorithm, such as `MINKNAP` [10], but `CPLEX` was fast enough for our purposes. Indeed, it solved all of the 0-1 KPs in less than one second.)

5.1. Test instances

Kataoka and Yamada [4] constructed some MKAP instances with random profits, weights and knapsack capacities. In these instances, the weights are random integers distributed uniformly between 1 and 1000. Their 'small' instances have $m \in \{10, 20\}$, $n \in \{20, 40, 60\}$, and $r \in \{2, 5\}$, whereas their 'large' instances have $m \in \{200, 400, 800\}$, $n \in \{4000, 8000\}$, and $r \in \{50, 100\}$. There is also a fourth parameter, which controls the degree of correlation between the profits and weights. The parameter takes three values: uncorrelated, weakly correlated and strongly correlated. We omit details, for brevity.

There are 10 instances for each combination of parameters, making a total of 360 small instances and 360 large instances. In all of these instances, $n_k = n/r$ for all $k \in R$, and the knapsack capacities are uniformly distributed between 0 and W/m, where W is the sum of the weights. (This means that the expected value of C is $W/2$.)

With the help of `CPLEX`, we were able to obtain the optimal solution values for the instances with $n \in \{20, 40\}$. Most of the larger instances, however, could not be solved to optimality within our chosen time limit (1200 seconds). Moreover, even for the instances with $n = 40$, we had to adjust the tolerances in `CPLEX` to obtain the true optimal values.

Some more MKAP instances were created by Martello and Monaci [7]. We omit the details, however, since we were unable to find optimal solution values for most of them.

In all of the above-mentioned instances, m is a multiple of r. For reasons which will become clear, we created some new instances that do not have this restriction. The profits and weights in our instances were created using the procedure from [4], but the instances have $m \in \{5, 15\}$, $n \in \{50, 100\}$ and $r = 10$. We set the cardinality of each class to n/r, as in [4], but we set the capacity of each knapsack to $W/2m$. We created ten random instances for each value of m and n, and each value of the correlation parameter, making 120 instances in total.

5.2. Results for some small Kataoka-Yamada instances

As mentioned above, we were able to obtain the optimal solution values for the instances with $n \in \{20, 40\}$. For each of those instances, we computed the four upper bounds U_{KY}, U_{MM}, U_{MM}^- and U_{GL}, and recorded the running times. We also computed the gap between each bound and the optimum, expressed as a percentage of the optimum.

The results for $n = 20$ and $n = 40$ are shown in Tables 1 and 2, respectively. The first three columns show the instance type. The next four columns report the average percentage gaps for each of the four bounds. The last two columns report the average computing time, in seconds, for U_{MM}^- and U_{GL}. (The times for the other two bounds are not reported, since they were less than one-hundredth of a second in every case.) Each figure is the average over the ten instances of the given type.

As one would expect from the theoretical results, $U_{KY} \geq U_{MM} \geq U_{MM}^- \geq U_{GL}$ in all cases. The difference between U_{KY} and U_{MM} is more marked when $n = 40$, whereas the difference between U_{MM} and U_{MM}^- is more marked when $n = 20$. Moreover, all bounds perform extremely poorly when $n = m = 20$.

We also see that, for these instances, U_{GL} is only a little better than U_{MM}^-. It also takes a bit longer to compute, but the running times are well under a second for all instances.

5.3. Results for new instances

Now, recall that our new instances have $n \in \{50, 100\}$, $m \in \{5, 15\}$ and $r = 10$. Table 3 reports the average percentage gaps and times for these instances. As before, each figure is the average over ten random instances.

The picture here is strikingly different to what we saw in the previous subsection. The new bound, U_{GL}, is much stronger than all of the other bounds, and it is even equal to the optimal value for all of the instances with $m = 5$.

The explanation for this phenomenon is as follows. When all item profits and weights are selected independently at random from the same distribution, the solution to the LP relaxation tends to "share out" the available capacity roughly equally among the r classes. The same is true for the Martello-Monaci KP relaxation (with or without reduction of the knapsack capacities). This behaviour is fine when m is a multiple of r, but unrealistic in other cases. Our bounding procedure avoids this weakness, by insisting that the amount of capacity allocated to each class belongs to the set Ω.

6. Conclusion

We have analysed some existing upper-bounding procedures for the MKAP, and proposed a new one. The new procedure turns out to be most useful when the number of knapsacks is not a multiple of the number of item classes.

As well as being another small step forward in exact algorithms for the MKAP, our work suggests that one must take care when constructing collections of benchmark instances for combinatorial optimisation problems. Indeed, we have seen that a small change in the instance parameters can lead to a huge change in the relative performance of different bounding procedures.

CRediT authorship contribution statement

Laura Galli: Conceptualization, Data curation, Software, Writing – review & editing. **Adam N. Letchford:** Methodology, Writing – original draft, Writing – review & editing.

Acknowledgement

The authors thank Silvano Martello for providing us with the benchmark MKAP instances. Thanks are also due to an anonymous referee.

Table 1
Average percentage gaps and times for the KY instances with $n = 20$.

			% gaps				time (s)	
Correlation	m	r	KY	MM	MM$^-$	GL	MM$^-$	GL
None	10	2	8.89	7.37	5.76	5.76	0.007	0.018
	10	5	10.68	9.14	7.06	6.01	0.006	0.031
	20	2	79.74	77.23	67.42	67.42	0.007	0.056
	20	5	79.74	77.23	62.84	62.29	0.008	0.013
Weak	10	2	11.14	9.76	7.27	7.21	0.011	0.016
	10	5	13.85	12.42	8.39	7.03	0.011	0.026
	20	2	110.13	107.62	90.75	90.75	0.011	0.050
	20	5	110.13	107.62	85.76	85.10	0.010	0.092
Strong	10	2	11.79	10.49	7.07	7.03	0.019	0.016
	10	5	14.22	12.90	6.09	5.14	0.012	0.023
	20	2	99.87	97.55	79.36	79.36	0.010	0.056
	20	5	99.87	97.55	73.62	73.57	0.010	0.115

Table 2
Average percentage gaps and times for the KY instances with $n = 40$.

			% gaps				time (s)	
Correlation	m	r	KY	MM	MM$^-$	GL	MM$^-$	GL
None	10	2	1.22	0.69	0.62	0.58	0.008	0.039
	10	5	2.81	2.27	2.03	1.60	0.008	0.088
	20	2	6.86	6.33	5.87	5.87	0.008	0.231
	20	5	7.37	6.83	5.92	5.92	0.009	0.678
Weak	10	2	1.13	0.75	0.58	0.57	0.008	0.040
	10	5	3.32	2.93	2.61	2.06	0.006	0.086
	20	2	7.49	7.02	6.37	6.37	0.008	0.304
	20	5	8.49	8.01	6.67	6.63	0.006	0.714
Strong	10	2	1.20	0.59	0.27	0.26	0.009	0.034
	10	5	2.75	2.13	1.63	1.23	0.009	0.091
	20	2	4.50	3.73	3.31	3.31	0.011	0.293
	20	5	5.27	4.51	3.34	3.34	0.010	0.702

Table 3
Average percentage gaps and times for new instances.

			% gaps				time (s)	
Correlation	n	m	KY	MM	MM$^-$	GL	MM$^-$	GL
None	50	5	42.47	42.03	41.63	0.00	0.012	0.005
	50	15	18.22	17.82	17.28	5.05	0.011	0.09
	100	5	48.96	48.78	48.73	0.00	0.009	0.011
	100	15	5.52	5.38	5.38	0.22	0.011	0.012
Weak	50	5	28.10	27.81	27.06	0.00	0.014	0.006
	50	15	19.33	19.07	18.42	7.09	0.013	0.009
	100	5	25.09	25.01	24.93	0.00	0.011	0.018
	100	15	5.28	5.22	5.21	0.53	0.012	0.014
Strong	50	5	19.60	18.82	17.91	0.00	0.015	0.005
	50	15	14.45	13.72	12.98	4.79	0.011	0.008
	100	5	15.89	15.53	15.50	0.00	0.010	0.014
	100	15	3.65	3.33	3.33	0.27	0.012	0.016

References

[1] E. Balas, E. Zemel, An algorithm for large zero-one knapsack problems, Oper. Res. 28 (1980) 1130–1154.
[2] R.E. Bellman, Dynamic Programming, Princeton University Press, Princeton, NJ, 1957.
[3] N.B. Dimitrov, D. Solow, J. Szmerekovsky, J. Guo, Emergency relocation of items using single trips: special cases of the multiple knapsack assignment problem, Eur. J. Oper. Res. 258 (2017) 938–942.
[4] S. Kataoka, T. Yamada, Upper and lower bounding procedures for the multiple knapsack assignment problem, Eur. J. Oper. Res. 237 (2014) 440–447.
[5] H. Kellerer, U. Pferschy, D. Pisinger, Knapsack Problems, Springer, Berlin, 2004.
[6] E. Lalla-Ruiz, S. Voß, A biased random-key genetic algorithm for the multiple knapsack assignment problem, in: C. Dhaenens, L. Jourdan, M.-E. Marmion (Eds.), Proc. LION '15, Springer, Cham, 2015, pp. 218–222.
[7] S. Martello, M. Monaci, Algorithmic approaches to the multiple knapsack assignment problem, Omega 90 (2020), article 102004.
[8] S. Martello, P. Toth, Knapsack Problems: Algorithms and Computer Implementations, Wiley, Chichester, 1990.
[9] M. Patriksson, A survey on the continuous nonlinear resource allocation problem, Eur. J. Oper. Res. 185 (2008) 1–46.
[10] D. Pisinger, A minimal algorithm for the 0-1 knapsack problem, Oper. Res. 45 (1997) 758–767.
[11] L. Zhang, S. Geng, The complexity of the 0/1 multi-knapsack problem, J. Comput. Sci. Technol. 1 (1986) 46–50.