Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 439609, 11 pages
http://dx.doi.org/10.1155/2015/439609

Research Article

Solving the Multiscenario Max-Min Knapsack Problem Exactly with Column Generation and Branch-and-Bound

Telmo Pinto, Cláudio Alves, Raïd Mansi, and José Valério de Carvalho

Centro de Investigação Algoritmi da Universidade do Minho, Escola de Engenharia, Universidade do Minho, 4710-057 Braga, Portugal

Correspondence should be addressed to Cláudio Alves; claudio@dps.uminho.pt

Received 25 November 2014; Revised 21 January 2015; Accepted 27 January 2015

Academic Editor: Lu Zhen

Despite other variants of the standard knapsack problem, very few solution approaches have been devised for the multiscenario max-min knapsack problem. The problem consists in finding the subset of items whose total profit is maximized under the worst possible scenario. In this paper, we describe an exact solution method based on column generation and branch-and-bound for this problem. Our approach relies on a reformulation of the standard compact integer programming model based on the Dantzig-Wolfe decomposition principle. The resulting model is potentially stronger than the original one since the corresponding pricing subproblem does not have the integrality property. The details of the reformulation are presented and analysed together with those concerning the column generation and branch-and-bound procedures. To evaluate the performance of our algorithm, we conducted extensive computational experiments on large scale benchmark instances, and we compared our results with other state-of-the-art approaches under similar circumstances. We focused in particular on different relevant aspects that allow an objective evaluation of the efficacy of our approach. From different standpoints, the branch-and-price algorithm proved to outperform the other state-of-the-art methods described so far in the literature.

1. Introduction

The multiscenario max-min knapsack problem is a variant of the well-known knapsack problem. The problem is characterized by a set of items with a given weight and profits, and by a knapsack whose capacity determines the unique constraint that applies. In this context, a scenario is defined through the set of profits that apply to each item, respectively. Hence, the profit of an item depends directly on the scenario that is considered. The objective of the multiscenario max-min knapsack problem is to determine the subset of items whose total weight is smaller than or equal to the knapsack capacity, and whose total profit is maximized in the worst scenario over all the possible scenarios.

The standard knapsack problem is a special case of the multiscenario max-min knapsack problem in which there is only one scenario. In this case, the worst scenario is always the only one that applies, and the max-min knapsack problem reduces to the problem of finding the items with the maximum total profit under this scenario. As a consequence, the multiscenario max-min knapsack problem is NP-hard. In fact, Yu showed in [1] that the problem is pseudopolynomially solvable when the number of scenarios is bounded, while it is strongly NP-hard when this number is unbounded. The complexity of min-max combinatorial optimization problems was further studied in [2] where different approximation results are provided. They proved in particular that the min-max regret knapsack problem is not at all approximable even in the case of two scenarios, and that the problem is also strongly NP-hard for a nonconstant number of scenarios. Specific variants of the min-max knapsack problem were also explored in [3] for the case where the item sizes are all equal to 1, and one has to choose a given number of items so as to minimize the total cost under the existing scenarios. In [3], Kasperski et al. showed that the problem is not approximable within a constant factor unless P = NP.

Unlike the standard knapsack problem and other variants which have been explored in depth in the literature [4, 5], the multiscenario max-min knapsack problem has received much less attention, and only recently different approaches began to be explored [1, 6–10]. In practice, solving large scale instances of the multiscenario max-min knapsack problem

up to optimality remains a challenge. In this paper, we describe and analyse an exact solution approach for the problem that relies on the combination of branch-and-bound and column generation. To the best of our knowledge, this is the first time that these methods are used together to solve this particular problem.

The first contributions towards the solution of the multiscenario max-min knapsack problem are due to Yu [1] and Iida [6]. Yu [1] proposed and analysed lower and upper bounds for the problem computed through surrogate relaxation. He described an exact branch-and-bound algorithm for the problem which solved instances with up to 60 items and 30 scenarios. Different lower and upper bounds based on linear programming were proposed later by Iida in [6]. To evaluate their effectiveness, the author embedded the bounds on the branch-and-bound algorithm proposed by Yu [1]. Although Iida claimed that the bounds were not as tight as those proposed by Yu [1], his computational experiments showed that the bounds were sufficient to solve the instances used by Yu up to optimality in less than one minute on average.

Taniguchi et al. [7] explored the use of the pegging test to reduce the size of the problem. The pegging test is applied after lower and upper bounds have been computed through surrogate relaxation. The optimal solution of the multiscenario max-min knapsack problem is then searched by applying branch-and-bound on the reduced problem. The authors report on computational experiments conducted on instances generated as described in [1]. The size of the instances goes up to 30 scenarios and 1000 items, although the method frequently found much difficulty in solving the largest instances up to optimality within the time limit of 1200 seconds. The authors also compared their approach with those by Yu [1] and Iida [6] on smaller instances with 60 items and up to 30 scenarios and concluded that it outperformed them both.

The same authors extended their work in [11] focusing on the two-scenario max-min knapsack problem. They described a heuristic algorithm with which they found solutions for instances with up to 16000 items. In their approach, lower and upper bounds are still obtained through surrogate relaxation, while the pegging test is used to reduce the size of the instance. The authors also describe a method to further decrease the size of the problem based on a so-called virtual pegging test.

In [8], a cooperative approach based on tabu search and combining two local search algorithms is proposed. The author described a generalized local search procedure which is used to diversify the search and a restricted local search procedure which is used to intensify the search in a given region. To generate initial feasible solutions for the problem, the author resorted to an iterative greedy heuristic. The computational experiments reported in [8] showed that good approximate solutions can be found with this approach within small computing times.

In [9], Hanafi et al. explored a hybrid approach to solve the max-min knapsack problem with two scenarios. Their approach is based on the iterative improvement of lower and upper bounds obtained through relaxations of mixed integer programming models, temporary variable fixing, and by enforcing pseudocuts that exclude prior solutions of the problem. The authors reported on computational experiments using instances with up to 20000 items which showed a better performance on correlated instances of the max-min knapsack problem with two scenarios.

More recently, Song et al. [10] proposed a fast heuristic to solve the multiscenario max-min knapsack problem. Their approach remains based on the solution of a surrogate relaxation of the original problem obtained by applying a subgradient algorithm. The authors explored different incomplete m-exchange algorithms consisting in swapping the values of some of the binary variables of the problem. Their approach showed to be able to generate good approximate solutions for large scale instances within reasonable time limits although its effectiveness seems to decrease with strongly correlated instances.

In this paper, we explore a novel approach for the exact resolution of the multiscenario max-min knapsack problem based on column generation and branch-and-bound. Prior preliminary results on the use of column generation to compute lower and upper bounds were discussed in [12] for the problem with only two scenarios. Here, we extend this approach by generalizing it to the case of multiple scenarios, and we propose a branch-and-price algorithm to search for optimal integer solutions. The performance of the global approach is evaluated through extensive computational experiments performed on large scale benchmark instances from the literature. The results illustrate the effectiveness of the branch-and-price algorithm in finding optimal solutions within reasonable computing times even for strongly correlated instances. Branch-and-price was never used before to solve exactly the multiscenario max-min knapsack problem. However, it is important to note that there are already some successful attempts reported in the literature concerning the application of column generation to other variants of the standard knapsack problem. Such an example can be found in [13] for the multiple-choice knapsack problem.

The outline of the paper is as follows. In Section 2, we describe formally the elements that characterize the multiscenario max-min knapsack problem, and we introduce the notation that will be used throughout the paper. In Section 3, we describe two integer programming formulations for the problem: a standard compact formulation and a column generation reformulation that relies on a Dantzig-Wolfe decomposition of the previous model. The components of our branch-and-price algorithm are described in Section 4. In Section 5, we report on the computational experiments performed to evaluate the performance of our approach. Some conclusions are finally drawn in Section 6.

2. The Multiscenario Max-Min Knapsack Problem

As referred to above, the multiscenario max-min knapsack problem is a generalization of the well-known standard knapsack problem. In the latter, one is given a set of items characterized by a weight w_i and a profit c_i. The objective

of this standard problem is to find the subset of items with the maximum total profit that fit into a given knapsack of capacity $\overline{w}$; that is, whose total weight is smaller than or equal to $\overline{w}$. Let n be the total number of items. The standard knapsack problem can be formulated using the following integer programming model:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} c_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} w_i x_i \leq \overline{w}, \\ & x_i \in \{0,1\}, \quad i = 1, \ldots, n, \end{aligned} \tag{1}$$

with the binary variables x_i indicating whether the item i is selected or not.

The max-min knapsack problem shares most of the defining characteristics of the standard knapsack problem. It is defined from a set of n items of weight w_i, $i = 1, \ldots, n$, and from a knapsack of capacity $\overline{w}$. The difference lies in the definition of the profits associated with the items. In the max-min knapsack problem, the profit of an item depends on the particular scenario that is considered. For a given scenario s, the profit of a given item i, $i = 1, \ldots, n$, under this scenario will be denoted by c_i^s. Hence, a scenario is defined as a set of profits that apply to the items of the problem. On the contrary, the weights of the items are independent from the scenarios, and remain equal to w_i for an item i whatever the scenario that may apply. Throughout the paper, we will denote by m the total number of scenarios. In the general case where the number m of scenarios is unbounded, the problem is referred to as the multiscenario max-min knapsack problem. In this context, the objective of the problem consists in finding the subset of items that fit in the knapsack and with the maximum total profit under the worst scenario, that is, under the scenario with the minimum total profit over all the scenarios. In the case where $m = 1$, the problem reduces to the standard knapsack problem, and, as a consequence, the latter can be considered as a special case of the multiscenario max-min knapsack problem.

In this paper, we will address the general multiscenario max-min knapsack problem where the number m of scenarios is unbounded. Furthermore, we will assume that all the items have positive integer profits, a weight which is smaller than or equal to the capacity $\overline{w}$ of the knapsack, and such that the items do not fit all in the knapsack, that is, $\sum_{i=1}^{n} w_i > \overline{w}$.

3. Integer Programming Formulations

3.1. The Standard Formulation. The multiscenario max-min knapsack problem can be formulated as follows:

$$\begin{aligned} \max \quad & \min_{s=1,\ldots,m} \left\{ \sum_{i=1}^{n} c_i^s x_i \right\} \\ \text{s.t.} \quad & \sum_{i=1}^{n} w_i x_i \leq \overline{w}, \\ & x_i \in \{0,1\}, \quad i = 1, \ldots, n, \end{aligned} \tag{2}$$

where the binary variables x_i represent the selection or not of an item i, $i = 1, \ldots, n$. This model is a direct extension of the integer programming model (1) for the standard knapsack problem.

The minimization part of the objective function can be reformulated using a set of equivalent constraints stating that the total profits under every scenario must be greater than or equal to a given variable z. The resulting model becomes a single-objective problem as illustrated next:

$$\max \quad z \tag{3}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} c_i^1 x_i \geq z, \tag{4}$$

$$\sum_{i=1}^{n} c_i^2 x_i \geq z, \tag{5}$$

$$\vdots \tag{6}$$

$$\sum_{i=1}^{n} c_i^m x_i \geq z, \tag{7}$$

$$\sum_{i=1}^{n} w_i x_i \leq \overline{w}, \tag{8}$$

$$x_i \in \{0,1\}, \quad i = 1, \ldots, n. \tag{9}$$

In [9], Hanafi et al. explored different mixed integer programming reformulations of the two-scenario max-min knapsack problem, and described different approaches to convert the multiobjective formulation (2) into a single-objective one.

3.2. A Column Generation Reformulation. A stronger model for the multiscenario max-min knapsack problem can be obtained from (3)–(9) by applying an appropriate Dantzig-Wolfe decomposition. The result is a column generation reformulation of (3)–(9) which is the base of the solution approach described in this paper.

Applying the Dantzig-Wolfe decomposition principle to a linear integer problem leads to a reformulation into a master problem and one or more subproblems defined from the constraints of the original formulation. Here, we consider a reformulation of (3)–(9) in which the master problem is defined from the constraints (4)–(7), and the subproblem from the knapsack constraints (8) and (9). Since the subproblem has not the integrality property, the resulting model is potentially stronger than the original formulation (3)–(9).

Let X denote the polyhedron given by the knapsack constraint (8), and let t be the number of extreme points of X. Since the polyhedron X is bounded, a point $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in X$ can be expressed as a convex combination of the extreme points of X. Let

$\mathbf{E_p} = (E_p^1, E_p^2, E_p^3, \ldots, E_p^n)$, $p = 1, \ldots, t$, be the pth extreme point of X such that

$$(x_1, x_2, \ldots, x_n) = \sum_{p=1}^{t} \lambda_p \mathbf{E_p} \tag{10}$$

$$= \left(\sum_{p=1}^{t} \lambda_p E_p^1, \sum_{p=1}^{t} \lambda_p E_p^2, \ldots, \sum_{p=1}^{t} \lambda_p E_p^n \right), \tag{11}$$

$$\sum_{p=1}^{t} \lambda_p = 1, \tag{12}$$

$$\lambda_p \geq 0, \quad p = 1, \ldots, t. \tag{13}$$

The master problem is further defined by replacing the variables $(x_1, x_2, \ldots, x_n)$ of (3)–(9) by the previous reformulation (11)–(13) of the knapsack polytope related to (8). The master problem states formally as follows:

$$\begin{aligned}
\max \quad & z \\
\text{s.t.} \quad & \sum_{i=1}^{n} c_i^1 \left(\sum_{p=1}^{t} \lambda_p E_p^i \right) \geq z, \\
& \sum_{i=1}^{n} c_i^2 \left(\sum_{p=1}^{t} \lambda_p E_p^i \right) \geq z, \\
& \vdots \\
& \sum_{i=1}^{n} c_i^m \left(\sum_{p=1}^{t} \lambda_p E_p^i \right) \geq z, \\
& \sum_{p=1}^{t} \lambda_p = 1, \\
& \lambda_p \geq 0, \quad p = 1, \ldots, t,
\end{aligned} \tag{14}$$

or equivalently

$$\max \quad z \tag{15}$$

$$\text{s.t.} \quad \sum_{p=1}^{t} \lambda_p \left(\sum_{i=1}^{n} c_i^1 E_p^i \right) \geq z, \tag{16}$$

$$\sum_{p=1}^{t} \lambda_p \left(\sum_{i=1}^{n} c_i^2 E_p^i \right) \geq z, \tag{17}$$

$$\vdots \tag{18}$$

$$\sum_{p=1}^{t} \lambda_p \left(\sum_{i=1}^{n} c_i^m E_p^i \right) \geq z, \tag{19}$$

$$\sum_{p=1}^{t} \lambda_p = 1, \tag{20}$$

$$\lambda_p \geq 0, \quad p = 1, \ldots, t. \tag{21}$$

The number of variables λ_p, $p = 1, \ldots, t$, of the master problem is exponential, as is the number t of extreme points of X. Constraint (20) corresponds to the convexity constraint (12). An upper bound for the multiscenario max-min knapsack problem can be computed from the linear relaxation of the master problem. This bound is greater than or equal to the bound computed from the linear relaxation of (3)–(9).

In a column generation approach where the master problem is solved iteratively by considering only a restricted subset of its variables, the subproblem defined from the constraints (8) and (9) of the original problem is used to price out the variables (columns) that are not in the restricted master problem and that may eventually improve its solution. The solutions provided by the subproblem are in fact the extreme points $\mathbf{E_p}$ of the knapsack polytope related to the constraints (8) and (9) of the original formulation.

Let $\pi_1, \pi_2, \ldots, \pi_m, \pi_0$ denote, respectively, the dual variables related to the constraints (16), (17), …, (19), and (20) of the master problem. A variable λ_p, $p = 1, \ldots, t$, of the master problem is attractive if and only if its reduced cost

$$\begin{aligned}
& -\pi_1 \left(\sum_{i=1}^{n} c_i^1 E_p^i \right) - \pi_2 \left(\sum_{i=1}^{n} c_i^2 E_p^i \right) - \cdots \\
& -\pi_m \left(\sum_{i=1}^{n} c_i^m E_p^i \right) - \pi_0
\end{aligned} \tag{22}$$

is positive. The most attractive column corresponds to the extreme point of X that maximizes (22). As a consequence, the pricing subproblem related to the Dantzig-Wolfe decomposition described in this section states formally as follows:

$$\min \quad \sum_{i=1}^{n} \left(\pi_1 c_i^1 + \pi_2 c_i^2 + \cdots + \pi_m c_i^m \right) y_i \tag{23}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i y_i \leq \overline{w}, \tag{24}$$

$$y_i \in \{0, 1\}, \quad i = 1, \ldots, n, \tag{25}$$

where y_i, $i = 1, \ldots, n$ are the decision variables of this standard knapsack problem.

Note that (23)–(25) provides a lower bound for the multiscenario max-min knapsack problem since it generates solutions which are feasible for this problem. Hence, a column generation algorithm applied to the reformulation discussed in this section generates at each iteration not only an upper bound from the solutions of the master problem, but also a lower bound from the solutions of the subproblem.

4. A Branch-and-Price Algorithm

As alluded above, for any medium or large size instance, the size of the model (15)–(21) prevents its resolution based on a complete enumeration of the variables (columns). As an alternative, we will consider an iterative resolution of a restricted master problem defined from (15)–(21) where only

a subset of the variables is used. Columns that are not in the restricted master problem, but which may potentially improve its solution are added on the fly using a column generation procedure.

The solution of (15)–(21) provides an upper (continuous) bound for the value of the optimal integer solution of the problem. Since the columns of the restricted master problem required to find the optimal solution of (15)–(21) may not be enough to find the optimal integer solution of the problem, we will resort to branch-and-bound where at the nodes of the search tree, the column generation procedure is used to find attractive columns that may not have been generated yet. This approach results in a so-called branch-and-price algorithm. The details of our branch-and-price algorithm are described in the sequel.

4.1. Column Generation Procedure. In our implementation, the restricted master problem is initialized with a limited subset of the variables of (15)–(21). A simple approach to generate an initial set of columns is to choose the solutions with only one item, and to add the corresponding columns to the restricted master problem (by assumption, all these solutions will satisfy the constraint on the capacity of the knapsack). The restricted master problem is then solved up to optimality.

The values of the dual variables related to the optimal solution of the restricted master problem are used as an input for the pricing subproblem. A set T of integer and best solutions for the subproblem is then computed using a dynamic programming algorithm. The maximum size of T is set through a parameter $t_{\max}$. Let z^s_{SP} denote the value of a solution $s \in T$. If $-z^s_{SP} - \pi_0$ is greater than 0, then the column related to this solution is attractive. Note that here we will consider that the set T includes only solutions that correspond to attractive columns. Hence, in practice, the size of T may be smaller than $t_{\max}$. The columns in T are added to the restricted master which is solved once again. Meanwhile, the set T is used to update (eventually) the value of the best lower bound for the problem. The process stops when there are no more solutions of the subproblem such that $-z^s_{SP} - \pi_0$ is greater than 0.

The optimal solution of the restricted master problem provides an upper bound for the value of the optimal integer solution of the problem, while the solutions generated by the pricing subproblems may improve the value of the lower bound for the problem as referred to above. Hence, our column generation procedure can be seen as a bounding algorithm for the multiscenario max-min knapsack problem.

In our implementation, instead of generating only one attractive column at each iteration which is the most usual approach in column generation, we generate a set of different attractive columns with the objective of visiting a significant number of feasible solutions for the problem. Note that each solution of the subproblem is feasible for the original multiscenario max-min knapsack problem. In our computational experiments, this strategy proved to be effective in the search for good incumbents.

Algorithm 1 describes formally our column generation procedure.

Algorithm 1 does not ensure that an optimal integer solution for the problem is found. At the end of the column generation procedure, the optimality gap may still be greater than 0. To overcome this issue, we resort to a branch-and-bound procedure which is described in the next section.

4.2. Branch-and-Bound. The branch-and-bound algorithm devised in this section is an exact algorithm that ensures that an optimal integer solution of the multiscenario max-min knapsack problem is found. The efficiency of branch-and-bound approaches depends critically on two main issues: the bounding strategy used to compute lower and upper bounds for the problem, and the branching strategy. By focusing on tightening the optimality gap, the former allows to reduce the size of the search tree, while the latter is used to guide the search so that the bounding strategy can be more effective.

In our algorithm, upper and lower bounds are computed using the column generation procedure described in the previous section (Algorithm 1). For the branching part of the algorithm, we exploit the information provided at each node of the search tree by the last pricing subproblem defined within the column generation process.

Our branch-and-bound algorithm is described formally in Algorithm 2. The algorithm is based on a LIFO (Last In, Fist Out) strategy. At each node of the search tree, the column generation algorithm (Algorithm 1) is used to solve the master problem (15)–(21) taking into account the set of variables fixed according to the branching strategy. After completion, we get an upper bound (ub_f) at the current node. Let $\boldsymbol{\pi}'$ denote the coefficients (profits) in the objective function (23) of the variables y_i in the last pricing subproblem (23)–(25) solved during the column generation process, and let $\mathbf{x}'$ be the solution of this last pricing subproblem. Note that we exclude from $\mathbf{x}'$ those variables whose value has been fixed during the branching process. Furthermore, let v^* be the value of the best solution of the multiscenario max-min knapsack problem found so far (incumbent solution).

Our branching scheme is based on the variables of the original compact formulation (3)–(9). In particular, the focus is put on the variables whose value has greater propensity to change in the optimal solution. From a heuristic standpoint, we consider that the ith variable is in this situation if it satisfies one of these two conditions:

(1) the value x'_i in $\mathbf{x}'$ is equal to 1, and its reduced cost (its approximation π'_i/w_i) is weak, that is, $i_f = \text{argmin}_i\{\pi'_i/w_i \text{ and } x'_i = 1\}$;

(2) the value of x'_i in $\mathbf{x}'$ is equal to 0, and its reduced cost (its approximation π'_i/w_i) is strong, that is, $i_f = \text{argmax}_i\{\pi'_i/w_i \text{ and } x'_i = 0\}$.

These two conditions translate the fact that the i^{th}_f variable may have more chances to change its values in the optimal solution if this one has not been found yet. The i^{th}_f variable is then selected for branching. It is fixed first to the value 1, while in the backtracking phase, a solution is sought for the case where its value is fixed to 0. These branching constraints are easily translated into the master problem (15)–(21), and

Let $(v^*, \mathbf{x}^*)$ denote the value of the best solution found, and the corresponding solution, respectively;
Let T denote the set of the best attractive solutions of the subproblem (with $|T| \leq t_{\max}$);
Initialization
  Initialize T with a set of solutions satisfying the knapsack constraint (24);
while $|T| > 0$ **do**
  Add the columns for the solutions in T to the restricted master problem related to (15)–(21);
  Solve the resulting restricted master problem;
  Define the subproblem (23)–(25) using the optimal dual values of the restricted master problem;
  Solve the resulting subproblem using dynamic programming;
  Update the set T with the attractive solutions found by solving the subproblem;
  Update the best solution $(v^*, \mathbf{x}^*)$;
end

Algorithm 1: Column generation algorithm for the multiscenario max-min knapsack problem.

Let v^* denote the value of the best solution found (incumbent);
Let $\mathbf{x}'$ denote the solution of the last pricing subproblem solved;
//We exclude from $\mathbf{x}'$ *those variables whose value has been fixed during the branching process.*
Let F be the list of indexes of the variables fixed to 0 or 1 in the branching tree;
Let f_{last} be the last index in the set F;
Initialization
  $F := \emptyset$; $init := true$;
while $F \neq \emptyset$ or $init = true$ **do**
  if $init = true$ **then** $init := false$;
  Solve the restricted master problem related to (15)–(21) with variables fixed according to F using Algorithm 1;
  Let ub_f be the value of the corresponding optimal solution;
  $backtracking := false$;
  if *the solution of the restricted master problem is integer* **then**
    $backtracking := true$;
    if $ub_f > v^*$ **then** $v^* := ub_f$;
  end
  else
    Let π' be the coefficients in (23) of the last pricing subproblem;
    $i_f := \arg\max_i \{\pi'_i / w_i \text{ and } x'_i = 0\}$;
    if $\nexists i_f$ **then** $i_f := \arg\min_i \{\pi'_i / w_i \text{ and } x'_i = 1\}$;
    if $\exists i_f$ and $ub_f > v^*$ **then** $F := F \cup \{i_f : x_{i_f} = 1\}$; // *Branch on the variable* x_{i_f};
    else $backtracking := true$;
  end
  if $backtracking = true$ **then**
    // *Backtracking*
    while $x_{f_{\text{last}}}$ *is fixed to 0* **do**
      $F := F \setminus f_{\text{last}}$;
    end
    if $F \neq \emptyset$ **then**
      Fix $x_{f_{\text{last}}}$ to 0;
      // $F := (F \setminus \{f_{\text{last}} : x_{f_{\text{last}}} = 1\}) \cup \{f_{\text{last}} : x_{f_{\text{last}}} = 0\}$
    end
  end
end

Algorithm 2: Branch-and-price algorithm for the multiscenario max-min knapsack problem.

they do not induce any additional complexity to the pricing subproblem (23)–(25). Backtracking is done whenever the upper bound ub_f is less than the best known value v^*. Finally, the branch-and-bound search stops when the search tree becomes empty.

5. Computational Experiments

To evaluate the performance of our branch-and-price algorithm, we conducted a set of extensive computational experiments on large scale benchmark instances of the literature.

In order to compare it objectively with current state-of-the-art methods, we divided our tests in two parts. In the first part, we report on results obtained using instances of the max-min knapsack problem with two scenarios. Our branch-and-price algorithm is evaluated from different standpoints, namely by considering a limit on the total computing time, by analysing the quality of the upper bounds obtained from it, and by analysing its potential in finding optimal integer solutions in reasonable amounts of computing time. In the first case, the focus is put on the capacity of the algorithm in generating good incumbents for the problem, as if the algorithm was being used as a heuristic. In the second part of our experiments, we will analyse the behavior of our approach in the general case where there are more than two scenarios. All the experiments were conducted on a PC with 2.4 GHz and 4 GB of RAM. Additionally, we used the version 11.1 of the CPLEX optimization solver.

5.1. Test Set I: Tests on Instances with Only Two Scenarios. The experiments reported in this section were conducted on the weakly and strongly correlated instances used in [9] which were obtained from the generator described in [11]. We used three sets of instances which are divided according to the capacity of the knapsack. The capacity $\overline{w}$ of the knapsack is set to $\sigma \times \sum_{i=1}^{n} w_i$, where $\sigma \in \{0.25, 0.5, 0.75\}$. For each value of n (number of items) and σ, there are 15 instances. For the set of weakly correlated instances, we have $n \in \{5000, 7000, 10000, 13000, 15000, 18000, 20000\}$, while for the strongly correlated instances, we have $n \in \{500, 1000, 4000, 5000, 7000, 10000\}$. Our results are compared with the exact resolution of the original model (3)–(9) through the commercial solver CPLEX, and with the approaches described by Taniguchi et al. in [11] and by Hanafi et al. in [9].

The presentation of the results is organized as follows. In the first part, we compare all these approaches on the aforementioned instances for a very limited execution time (5 seconds). The objective is to evaluate the capacity of each approach in finding good feasible solutions quickly, as if they were used as heuristics. In the second part, we compare the quality of the upper bounds given by our approach with those obtained with CPLEX and the approach of Taniguchi et al. [11]. Note that the algorithm described by Hanafi et al. [9] uses initially the same upper bounds as CPLEX before improving it by adding cuts during the execution of their algorithm. Finally, we compare the results of the different approaches for a larger execution time to evaluate the capacity of each one in finding proven optimal solutions for the problem.

5.1.1. Evaluating the Branch-and-Price Algorithm as a Heuristic. In this section, we evaluate the capacity of our branch-and-price algorithm in finding quickly good feasible solutions for the problem, and we compare it with the methods described by Taniguchi et al. [11] and by Hanafi et al. [9]. For this purpose, we used a time limit of 5 seconds. The results for instances described above are presented in Tables 1 and 2. The meaning of the corresponding columns is the following:

(i) n: number of items in the corresponding instance;

(ii) *#best*: number of times the corresponding algorithm found the best solution;

(iii) *cpu*: average computing time (in seconds) required to find the corresponding solution.

Table 1 shows that CPLEX provides the best solutions for an important number of cases, although this performance tends to decrease with the size of problem. The approach of Taniguchi et al. [11] exhibits the worst results among all the methods that were tested. The performance of the approach of Hanafi et al. [9] is comparable to the performance of our branch-and-price algorithm, although it tends to be better on these instances for larger values of σ. In fact, the algorithm of Hanafi et al. fixes the values of the variables by exploiting its improved upper and lower bounds. These fixing rules are useful when the correlation between the coefficients of the problem is weak. The fixation of an important number of variables in the algorithm of Hanafi et al. helps it to deal with this type of instances independently of the value of σ. On the contrary, this parameter has a non-negligible impact on our algorithm. When $\sigma = 0.25$, our algorithm is the best among all the methods, while it is the second best for $\sigma = 0.5$ and $\sigma = 0.75$. The value of σ defines the capacity of the knapsack. The capacity of the knapsack becomes larger as the value of σ increases, which impacts on the dynamic programming algorithm used to solve the pricing subproblems in column generation.

For the strongly correlated instances (Table 2), the performance of CPLEX when used directly on (3)–(9) tends to decrease with the size of the problem independently of the value of σ. The algorithm of Taniguchi et al. found much difficulty in finding the best solutions among all the algorithms even for the small size instances. This difficulty increases with the size of the problem, while it seems independent of the value of σ. Table 2 shows the algorithm of Hanafi et al. outperforms the last two approaches, while its performance decreases also with the size of the problem. The branch-and-price algorithm, whose results are provided in last columns of Table 2, clearly outperforms all the other approaches. The algorithm found all the best solutions in a small amount of time. The computing time increases with the size of the instance, but it remained almost always less than one second on average.

5.1.2. Quality of the Upper Bounds. In this section, we compare the upper bounds provided by the different approaches. The quality of the upper bounds is important to improve the convergence of branch-and-bound algorithms, and in methods that rely on fixing the values of the variables. Using the same instances as in the previous tests, we compare the upper bounds given by the linear relaxation of (3)–(9) (solved with CPLEX) with the algorithm of Taniguchi et al. [11] and Algorithm 1 applied to the master problem (15)–(21). We do not compare these upper bounds with those obtained with the method of Hanafi et al. [9] because the latter relies on the resolution of the model (3)–(9) and, as a consequence, it provides upper bounds which are similar to those reported for CPLEX.

Table 1: Comparative results on weakly correlated instances for a limited execution time (5 seconds).

n	σ	CPLEX		Taniguchi et al. [11]		Hanafi et al. [9]		Algorithm 2	
		#best	*cpu*	*#best*	*cpu*	*#best*	*cpu*	*#best*	*cpu*
5000	0,25	11	5,0	0	5,2	8	3,2	6	1,7
7000		6	5,0	1	4,8	6	2,4	5	2,4
10000		5	5,0	0	4,4	7	2,4	8	2,4
13000		4	5,0	0	3,2	9	2,5	8	1,9
15000		2	5,0	0	2,8	8	3,4	5	3,3
18000		3	5,0	0	2,4	4	3,6	11	2,7
20000		2	5,1	0	2,8	5	3,3	11	3,0
		33		*1*		*47*		*54*	
5000	0,50	7	5,0	1	5,6	6	4,2	5	2,1
7000		8	5,0	0	4,8	4	2,9	8	2,5
10000		4	5,0	2	4,0	6	2,1	5	2,9
13000		2	5,0	0	4,8	12	2,9	7	3,2
15000		5	4,9	0	4,4	9	3,5	5	2,6
18000		4	5,0	0	2,4	6	4,1	9	2,7
20000		2	5,0	0	4,0	5	3,7	8	2,6
		32		*3*		*48*		*47*	
5000	0,75	6	5,0	0	4,0	9	3,7	7	2,0
7000		6	4,9	0	4,8	5	2,9	4	3,0
10000		4	5,0	0	3,6	6	1,5	8	3,1
13000		4	5,0	0	3,6	9	2,8	3	2,3
15000		2	4,8	0	3,2	10	3,6	3	2,9
18000		2	5,1	0	3,2	8	3,1	5	2,3
20000		5	5,0	0	3,2	6	3,5	3	2,5
		29		*0*		*52*		*33*	

Table 2: Comparative results on strongly correlated instances for a limited execution time (5 seconds).

n	σ	CPLEX		Taniguchi et al. [11]		Hanafi et al. [9]		Algorithm 2	
		#best	*cpu*	*#best*	*cpu*	*#best*	*cpu*	*#best*	*cpu*
500	0,25	15	5,00	11	3,33	15	1,47	15	0,33
1000		13	5,00	9	3,60	14	2,00	15	0,07
4000		9	5,00	2	1,20	14	3,33	15	0,40
5000		9	5,00	1	0,40	15	3,47	15	0,47
7000		10	5,00	1	0,80	10	2,87	15	0,27
10000		11	5,00	2	1,60	8	3,67	15	0,67
		67		*26*		*74*		*90*	
500	0,5	15	5,00	10	3,93	15	1,73	15	0,00
1000		15	5,00	7	3,20	15	1,53	15	0,00
4000		14	5,00	1	0,40	14	2,00	15	0,33
5000		12	5,00	4	2,00	15	2,67	15	0,33
7000		11	5,00	1	0,40	8	2,20	15	0,33
10000		8	5,00	2	0,80	6	2,13	15	0,80
		75		*25*		*73*		*90*	
500	0,75	15	5,00	5	3,20	15	2,93	15	0,00
1000		14	5,00	2	1,60	15	1,00	15	0,00
4000		12	5,00	3	1,60	15	2,13	15	0,47
5000		14	5,00	2	0,80	14	2,47	15	0,40
7000		12	5,00	3	1,60	12	4,13	15	0,40
10000		8	5,00	5	2,00	9	1,87	15	1,20
		75		*20*		*80*		*90*	

Table 3: Comparative results: upper bounds.

		Weakly correlated instances					Strongly correlated instances		
n	σ	CPLEX	Taniguchi et al. [11]	Algorithm 1	n	σ	CPLEX	Taniguchi et al. [11]	Algorithm 1
		ub	*Gap*	*Gap*			*ub*	*Gap*	*Gap*
5000		896067,07	−2,11	0,55	500		87479,19	−0,15	21,59
7000		1253428,76	−3,31	0,36	1000		174658,07	−0,34	19,32
10000		1792894,59	−3,70	0,24	4000		700131,20	−0,59	28,04
13000	0,25	2331663,99	−4,53	0,20	5000	0,25	875128,10	−1,61	25,63
15000		2688442,51	−6,01	0,21	7000		1225569,44	−1,32	24,69
18000		3225076,77	−6,53	0,13	10000		1750641,34	−2,33	18,75
20000		3585803,96	−6,53	0,14					
5000		1632862,83	−2,09	0,46	500		160240,63	−1,41	19,90
7000		2284894,04	−3,15	0,34	1000		320830,19	−0,30	23,88
10000		3263829,38	−4,23	0,27	4000		1282512,32	−1,24	31,36
13000	0,50	4243497,29	−4,42	0,19	5000	0,50	1603836,11	−6,38	27,29
15000		4896680,33	−6,77	0,13	7000		2244474,61	−15,69	47,88
18000		5875562,14	−10,03	0,10	10000		3206448,08	−19,28	46,87
20000		6528236,22	−6,85	0,11					
5000		2335623,64	−2,00	0,51	500		230631,50	−1,19	18,10
7000		3268655,45	−3,26	0,37	1000		461424,33	−3,84	40,67
10000		4669024,62	−4,20	0,30	4000		1847076,46	−2,13	40,66
13000	0,75	6072679,02	−6,12	0,20	5000	0,75	2309220,20	−6,94	40,59
15000		7003993,61	−6,10	0,19	7000		3231105,62	−2,66	18,76
18000		8406040,02	−6,81	0,15	10000		4615654,05	−3,55	34,91
20000		9338818,70	−5,71	0,11					

Table 3 gives the average results for the weakly and strongly correlated instances. Column *ub* associated with CPLEX presents the average upper bound provided by CPLEX over the different groups of 15 instances. The columns *Gap* related to the algorithm of Taniguchi et al. and Algorithm 2 give the average absolute gap between their corresponding upper bound and the upper bound provided by CPLEX (*Gap* = CPLEX upper bound – upper bound provided by the corresponding algorithm). Hence, in Table 3, when *Gap* is positive, the upper bound of the corresponding algorithm is better than the one provided by CPLEX.

Table 3 shows that the bounds provided by our column generation algorithm are always better than those provided by the other two approaches. Recall that since our pricing subproblem has not the integrality property, the quality of the upper bounds provided by our reformulation (15)–(21) are potentially better than those obtained by using the compact model (3)–(9). Furthermore, the approach of Taniguchi et al. relies on a surrogate relaxation that is worse than the linear relaxation of (3)–(9). The comparative quality of the upper bounds provided by our column generation increases as the instances become more strongly correlated.

5.1.3. Evaluating the Branch-and-Price Algorithm as an Exact Algorithm. Here, we report on the results obtained with our branch-and-price algorithm on the strongly correlated instances with a time limit of 10 minutes. The objective is to evaluate the capacity of our algorithm in finding proven optimal solutions for the problem. The results are presented in Table 4. Column #*opt* gives the number of proven optimal solutions found for each set of 15 instances, while column *cpu* indicates average computing time required to reach the corresponding solution.

Our branch-and-price algorithm found the optimal solution for 181 instances out of 270 within the time limit, which represents more than 67% of the tested instances. In many cases, the solution is found in the first nodes of the branching tree, which is due to the quality of the bounds computed using our column generation approach. For the other cases where the optimality of the solution has not been proved, the optimality gap remains much smaller than the one obtained by using CPLEX with the same time limit. The other approaches from Taniguchi et al. [11] and Hanafi et al. [9] are also clearly outperformed by the branch-and-price algorithm described in this paper. Indeed, the approach of Taniguchi et al. solved only 3 of the 270 instances up to optimality, while the algorithm of Hanafi et al. solved exactly only 5 of these 270 instances.

5.2. Test Set II: Tests on Instances with More Than Two Scenarios. To evaluate the performance of our branch-and-price algorithm on general instances of the multiscenario max-min knapsack problem, we generated a set of instances using the generator proposed in [7]. In particular, to test the limits of our algorithm, we consider the case of strongly correlated instances with $\sigma = 0.75$. Recall that the capacity of the knapsack increases with σ. This increase has a non-negligible impact on the resolution of the pricing subproblem through

Table 4: Number of proven optimal solutions found by branch-and-price on strongly correlated instances.

n	σ	#opt	cpu	n	σ	#opt	cpu	n	σ	#opt	cpu
500	0,25	6	400,4	500	0,5	5	400,3	500	0,75	7	345,3
1000		6	360,4	1000		8	280,6	1000		10	200,6
4000		10	206,4	4000		11	160,7	4000		13	80,8
5000		12	120,8	5000		11	160,7	5000		14	66,5
7000		10	200,8	7000		12	120,8	7000		10	200,6
10000		10	204,0	10000		14	42,9	10000		12	120,8

Table 5: Comparative results over large scale instances with more than two scenarios (Algorithm 2 *versus* CPLEX).

Instances		60 seconds		300 seconds		600 seconds	
n	m	#best	#only best	#best	#only best	#best	#only best
1000	100	0	0	0	0	0	0
	500	10	10	10	10	10	10
	1000	10	10	10	10	10	10
5000	100	10	10	10	10	7	7
	500	10	10	10	10	10	10
	1000	10	10	10	10	10	10
10000	100	10	10	10	10	10	10
	500	10	10	10	10	10	10
	1000	10	10	10	10	10	10
20000	100	10	10	10	10	10	10
	500	10	10	10	10	10	10
	1000	10	10	10	10	10	10

dynamic programming. Furthermore, on these instances, the method of Taniguchi et al. [7] is outperformed by the exact resolution of (3)–(9) using CPLEX. Additionally, we consider large scale instances with $n \in \{1000, 5000, 10000, 20000\}$ and $m \in \{100, 500, 1000\}$ (number of scenarios). Within each set, we generated 10 instances. Our branch-and-price algorithm is compared with the exact resolution of (3)–(9) using CPLEX.

In Table 5, we show the results obtained using a time limit of 60, 300 and 600 seconds, respectively. In column (*#best*), we give the number of times where our algorithm provides the best solution when compared to CPLEX. The number of times in which our branch-and-price algorithm is the only one to provide the best solution is given in column (*#only best*).

While CPLEX provides better results for the set of smallest instances with $n = 1000$ and $m = 100$, it is clearly outperformed for the other 11 sets of larger instances. For example, as the number of items increases from $n = 1000$ to $n = 5000$ (with $m = 100$), CPLEX did not provide any best solution in 60 and 300 seconds, while it provided only 3 best solutions when it ran during 600 seconds. This behavior may be explained in part by the results presented in the previous sections, and in particular those that illustrate the quality of the lower and upper bounds provided by our branch-and-price algorithm. Indeed, the branch-and-price algorithm proved to be able to generate good incumbents quickly while relying at the same time on a stronger (column generation based) model from which strong upper bounds can be derived.

6. Conclusions

In this paper, we explored for the first time an exact solution approach for the multiscenario max-min knapsack problem based on column generation and branch-and-bound. The resulting branch-and-price algorithm proved to outperform in different aspects other state-of-the-art methods described in the literature. The column generation reformulation provides stronger upper bounds, while the corresponding subproblems can be used at the same time to price out attractive columns and to generate feasible solutions for the max-min knapsack problem that may contribute to improve the incumbent solution. Furthermore, the branching strategy explored in this paper is such that it does not induce any additional complexity to the pricing subproblem. Extensive computational experiments were conducted and discussed using large scale benchmark instances. Comparative results with other approaches were provided, supporting the evidence that branch-and-price performs globally better than the best known methods of the literature.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported by FEDER funding through the Programa Operacional Factores de Competitividade, COMPETE, and by national funding through the Portuguese Science and Technology Foundation (FCT) in the scope of the project PTDC/EGE-GES/116676/2010 (reference COMPETE: FCOMP-01-0124-FEDER-020430). Additionally, this work was supported by FCT through the doctoral grant SFRH/BD/73584/2010 for Telmo Pinto (funded by QREN, POPH, Typology 4.1, cofunded by MEC National Funding and the European Social Fund), and by FEDER funds through the Competitiveness Factors Operational Programme, COMPETE. We would like to thank the referee for the time he/she spent on the reviewing process involved in this paper, for his/her helpful comments, and for his/her recommendations. His/her suggestions have greatly improved the contents and the presentation of this paper.

References

[1] G. Yu, "On the max-min 0-1 knapsack problem with robust optimization applications," *Operations Research*, vol. 44, no. 2, pp. 407–415, 1996.

[2] H. Aissi, C. Bazgan, and D. Vanderpooten, "Approximation of min-max and min-max regret versions of some combinatorial optimization problems," *European Journal of Operational Research*, vol. 179, no. 2, pp. 281–290, 2007.

[3] A. Kasperski, A. Kurpisz, and P. Zielinski, "Approximating the min-max (regret) selecting items problem," *Information Processing Letters*, vol. 113, no. 1-2, pp. 23–29, 2013.

[4] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*, John Wiley & Sons, New York, NY, USA, 1990.

[5] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*, Springer, 2004.

[6] H. Iida, "A note on the max-min 0-1 knapsack problem," *Journal of Combinatorial Optimization*, vol. 3, no. 1, pp. 89–94, 1999.

[7] F. Taniguchi, T. Yamada, and S. Kataoka, "Heuristic and exact algorithms for the max-min optimization of the multi-scenario knapsack problem," *Computers and Operations Research*, vol. 35, no. 6, pp. 2034–2048, 2008.

[8] A. Sbihi, "A cooperative local search-based algorithm for the multiple-scenario max-min knapsack problem," *European Journal of Operational Research*, vol. 202, no. 2, pp. 339–346, 2010.

[9] S. Hanafi, R. Mansi, C. Wilbaut, and A. Fréville, "Hybrid approaches for the two-scenario max-min knapsack problem," *International Transactions in Operational Research*, vol. 19, no. 3, pp. 353–378, 2012.

[10] X. Song, R. Lewis, J. Thompson, and Y. Wu, "An incomplete m-exchange algorithm for solving the large-scale multi-scenario knapsack problem," *Computers & Operations Research*, vol. 39, no. 9, pp. 1988–2000, 2012.

[11] F. Taniguchi, T. Yamada, and S. Kataoka, "A virtual pegging approach to the max-min optimization of the bi-criteria knapsack problem," *International Journal of Computer Mathematics*, vol. 86, no. 5, pp. 779–793, 2009.

[12] C. Alves, R. Mansi, J. Valério de Carvalho, and T. Pinto, "A column generation approach for the bi-objective max-min knapsack problem," in *Proceedings of the 1st International Conference on Operations Research and Enterprise Systems (ICORES '12)*, pp. 165–170, Vilamoura, Portugal, February 2012.

[13] D. Pisinger, "Budgeting with bounded multiple-choice constraints," *European Journal of Operational Research*, vol. 129, no. 3, pp. 471–480, 2001.