Comput Optim Appl (2010) 46: 51–73
DOI 10.1007/s10589-008-9184-7

A column generation method for the multiple-choice multi-dimensional knapsack problem

N. Cherfi · M. Hifi

Received: 18 July 2006 / Revised: 25 April 2008 / Published online: 28 May 2008

Abstract In this paper, we propose to solve large-scale multiple-choice multi-dimensional knapsack problems. We investigate the use of the column generation and effective solution procedures. The method is in the spirit of well-known local search metaheuristics, in which the search process is composed of two complementary stages: (i) a rounding solution stage and (ii) a restricted exact solution procedure. The method is analyzed computationally on a set of problem instances of the literature and compared to the results reached by both Cplex solver and a recent reactive local search. For these instances, most of which cannot be solved to proven optimality in a reasonable runtime, the proposed method improves 21 out of 27.

Keywords Branch-and-bound · Column generation · Heuristics · Knapsack · Optimization

1 Introduction

Integer Linear Programming (ILP) plays a central role in modeling difficult-to-solve (NP-hard) combinatorial optimization problems (see [5, 10, 16]). However, the exact solution of the resulting models often cannot be realized for the problem sizes of interest in real-world applications and so, the availability of effective heuristic solution methods is of paramount importance.

In this paper we investigate the use of the column generation and effective heuristic solution procedures for solving a particular 0-1 knapsack problem (see [2, 4])

N. Cherfi
CES, Equipe CERMSEM, Maison des Sciences Economiques, Université Paris 1
Panthéon-Sorbonne, 106-112, boulevard de l'Hôpital, 75634 Paris Cedex 13, France

M. Hifi (✉)
MIS, Axe Discrete Optimization and Re-optimization, Université de Picardie Jules Verne,
5 rue du Moulin Neuf, 80000 Amiens, France
e-mail: hifi@u-picardie.fr

known as Multiple-choice Multi-dimensional Knapsack Problem (MMKP). MMKP concerns many practical problems in the real life as service level agreement or model of allocation resources or as a dynamic adaptation of system of resources for multimedia multi-sessions (for more details, one can refer to [11, 12]).

In the MMKP, we have a multi-constrained knapsack of a capacity vector or available resources, namely $R = (R^1,\ R^2, \dots, R^m)$, and a set $S = (S_1, \dots, S_i, \dots, S_n)$ of items divided into n disjoint classes, where each class $i,\ i = 1, \dots, n$, has $r_i = |S_i|$ items. Each item $j,\ j = 1, \dots, r_i$, of class i has a nonnegative profit value v_{ij}, and requires resources given by the weight vector $W_{ij} = (w_{ij}^1, w_{ij}^2, \dots, w_{ij}^m)$. Each weight component w_{ij}^k (with $1 \le k \le m,\ 1 \le i \le n,\ 1 \le j \le r_i$) also has a nonnegative value. The problem is to fill the knapsacks with exactly one item from each class in order to maximize the total profit value of the choice, such that the capacity constraints are satisfied. By the total profit value of the choice, we mean the sum of the profits of items fixed in the multi-constrained knapsack. The MMKP can be formulated as follows:

$$
\begin{aligned}
Z_{\mathrm{MMKP}} = \max \quad & \sum_{i=1}^{n}\sum_{j=1}^{r_i} v_{ij}x_{ij} \\
\text{s.t.} \quad & \sum_{i=1}^{n}\sum_{j=1}^{r_i} w_{ij}^k x_{ij} \le R^k, \quad && k \in \{1, \dots, m\}, \\
& \sum_{j=1}^{r_i} x_{ij} = 1, && i \in \{1, \dots, n\}, \\
& x_{ij} \in \{0, 1\}, && i \in \{1, \dots, n\},\ j \in \{1, \dots, r_i\},
\end{aligned}
$$

where x_{ij} is either 0, implying item j of the i-th class S_i is not picked, or 1 implying item j of the i-th class is picked.

The remainder of the paper is organized as follows. In Sect. 2, we present a brief reference of some solution procedures for the MMKP. The concept of the column generation is summarized in Sect. 3. The adaptation of the column generation used by the proposed algorithm is presented in Sect. 3.2. In Sect. 3.3, we describe the solution representation and how we obtain the starting solution for our column generation procedure. In Sect. 3.4 we present the framework of the proposed algorithm which can be viewed as a restricted branch-and-bound search procedure. Finally, in Sect. 4, the performance of both versions of the proposed algorithm is evaluated on a set of problem instances of the literature varying from small to large sized ones.

2 Related works

To our knowledge, very few papers dealing directly with the MMKP are available. Indeed, the first paper tackling the heuristic resolution of the problem is due to Moser et al. [14]. In the last paper, the authors have designed an approach based upon the concept of graceful degradation from the most valuable items based on Lagrange multipliers. Khan et al. [12] have tailored an algorithm based on the aggregate resources already introduced by Toyoda [15] for solving the 0-1 multidimensional knapsack problem. Hifi et al. [9] proposed a guided local search in which the trajectories of the solutions were oriented by increasing the cost function with a penalty term; it penalizes bad features of previously visited solutions. In [8] a reactive local search has been proposed in which both *deblocking* and *degrading* procedures are introduced in order (i) to escape to local optima and (ii) to introduce diversification to the feasible search space. Finally, in [1], a parallelization of Hifi et al.'s algorithm has been proposed in which the authors tried to accelerate the search process but without improving the quality of the solutions.

In this paper, we propose a special restricted branch-and-bound procedure in which some nodes (called elite nodes) are solved using a column generation solution procedure combined with a heuristic search procedure. The proposed approach can be summarized as follows:

1. Construct a *starting solution* to the original MMKP, namely F, and let ILP be the *restricted* integer linear problem containing the columns associated to F (see Sect. 3.3).
2. Let η be an *expended node* (representing the root node at the beginning) corresponding to the *restricted* ILP associated to MMKP. Then,
   (a) Apply the *column generation procedure* to the relaxed ILP corresponding to the selected node η.
   (b) Perform a *greedy rounding heuristic* on a part of variables of ILP (corresponding to η), noted I_1, and complete the second part, noted I_2, using a *specialized solution procedure*.
3. Create new nodes by using some *branching strategies* and remove the ancestor node η and all nodes whose objective value (upper bound—a maximization problem) is less than or equal to the best current feasible solution value.
4. Exit with the best solution if the stopping condition is verified or when the list of nodes is reduced to empty set; repeat steps 2-3 otherwise.

3 A column generation solution method

Herein, we first describe the main principle of the proposed column generation. Next, we present how to generate an initial solution for the MMKP. Then, we explain how to generate the initial columns for starting the column generation method. Later we explain the used branching which is in general necessary for solving the MMKP. Finally, we show how we can produce integer solutions for the MMKP using two alternatives approaches: (i) a greedy rounding solution procedure and (ii) a hybrid solution procedure combining some treatments of the rounding procedure and a restricted exact procedure.

3.1 The principle of the column generation

The column generation method, originally proposed by Gilmore and Gomory [6, 7] for the well-known cutting stock problem, is a decomposition technique for solving a structured linear program (LP) with few rows but a lot off columns.

Generally, the column generation procedure (CGP) in mainly based upon decomposing the original LP into a master problem and a subproblem. The master problem contains a first subset of the columns and the subproblem, which is a separation problem for the dual LP, is solved to identify whether the master problem should be enlarged with additional columns or not. CGP alternates between the master problem and the subproblem, until the former contains all the columns that are necessary for reaching an optimal solution of the original LP.

From a computational point of view, the formulation of the MMKP are not suitable to use, especially when large-scale problem instances are considered. In particular, the numbers of constraints and variables grow rapidly with respect to the number of classes and the number of items of each class. In our numerical experiments, a state-of-the-art solver (Cplex solver) could solve the MMKP with up to 50 classes containing 10 items each (representing 500 items) to optimality, but failed to provide optimal (or "good near-optimal") solutions for larger instances—in reasonable runtime.

3.2 Adaptation of CGP for the MMKP

CGP is especially attractive for problems that can be formulated using multiple knapsack formulations, which typically contain a large number of columns, although very few of them are used in the optimal solution. We note that such a method has been proposed by Mehrotra and Trick [13] to solve the graph coloring problem and later used by Björklund et al. [3] for solving the special time division multiple access scheduling problem. In both papers, the authors showed that such a method is able to efficiently solve large-scale problem instances. Herein, we propose a variant of the column generation solution procedure which is used as a solution procedure for a subset of nodes.

To apply the CGP to MMKP, we consider its LP-relaxation defined as follows:

$$
\begin{aligned}
Z^{\mathrm{LP}}_{\mathrm{MMKP}} = \max \quad & \sum_{i=1}^{n}\sum_{j=1}^{r_i} v_{ij}x_{ij} \\
\text{s.t.} \quad & \sum_{i=1}^{n}\sum_{j=1}^{r_i} w_{ij}^{k}x_{ij} \le R^k, \qquad k \in \{1,\dots,m\}, && (1)\\
& \sum_{j=1}^{r_i} x_{ij} = 1, \qquad i \in \{1,\dots,n\}, && (2)\\
& 0 \le x_{ij} \le 1, \qquad i \in \{1,\dots,n\},\ j \in \{1,\dots,r_i\}.
\end{aligned}
$$

The column generation master problem is the same as the above LP-relaxation, except that the set of columns S is replaced by a subset $S_0 = \bigcup_{i=1}^{n} S_{i0}$ ($S_{i0} \subseteq S_i$, $S_0 \subseteq S$) of columns. The problem can be described as follows:

$$Z^{\mathrm{LP}}_{\mathrm{MMKP}} = \max \sum_{i=1}^{n} \sum_{j\in S_{i0}} v_{ij}x_{ij}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} \sum_{j\in S_{i0}} w^k_{ij}x_{ij} \le R^k, \quad k \in \{1,\dots,m\}, \tag{3}$$

$$\sum_{j\in S_{i0}} x_{ij} = 1, \quad i \in \{1,\dots,n\}, \tag{4}$$

$$0 \le x_{ij} \le 1, \quad i \in \{1,\dots,n\},\ j \in S_{i0}.$$

To ensure the feasibility of the master problem, the subset of the selected columns must satisfy (1) and (2). As we will show later, one particular choice of the subset S_0 of the starting columns is the node provided by the starting solution described in Sect. 3.3.

Note that when the master problem is solved, we need to identify whether it can be improved by adding new columns to S_0. In LP terms, this amounts to examining whether there exists any item per class on the rest of the columns, for which the corresponding variable x_{ij} has a strict positive reduced cost.

Using LP-duality, the reduced cost c_{ij} of variable x_{ij} is

$$c_{ij} = v_{ij} - \sum_{k=1}^{m} \lambda_k w^k_{ij} - \gamma_i,$$

where λ_k, $k \in \{1,\dots,m\}$, and γ_i, $i \in \{1,\dots,n\}$, are the optimal dual variables associated to (3) and (4), respectively. The above equality can be easily obtained by solving the following optimization problem:

$$(SP) \quad \text{such that} \quad \max_{i\in\{1,\dots,n\}} \big\{Z(SP_i)\big\},$$

where $Z(SP_i)$ denotes the optimal solution of (SP_i) defined as follows:

$$(SP_i) \quad \begin{cases} \max & \displaystyle\sum_{j=1}^{r_i}\left(v_{ij} - \sum_{k=1}^{m} w^k_{ij}\lambda_k\right)x_{ij} - \gamma_i \\ \text{s.t.} & \displaystyle\sum_{j=1}^{r_i} x_{ij} = 1, \\ & x_{ij} \in \{0,1\} \quad j = 1,\dots,r_i. \end{cases}$$

Note that the problem SP_i, $i \in \{1,\dots,n\}$, is equivalent to the following problem (SP_i):

$$\max_{j\in S_i}\left\{ v_{ij} - \sum_{k=1}^{m} \lambda_k w^k_{ij} - \gamma_i \right\}.$$

Since solving the problems SP_i, $i = 1, \dots, n$, is not time consuming, then at each step of the column generation procedure, we introduce the best n columns corresponding to the best reduced costs of each problem SP_i.

3.3 The starting solution/columns

Herein, we present a constructive procedure which permits to produce a starting solution. In fact, the purpose of this procedure is twofold: (i) constructing an initial solution for curtailing the search process and (ii) initializing the column generation procedure with the columns (and other ones) of the provided solution.

The initial solution is obtained by applying a *constructive procedure* (CP) already proposed in [9]. Observe that finding a feasible solution to MMKP remains a difficult task (i.e., NP-hard) and so, CP—a greedy procedure—tries to reach a feasible solution using a two-phase construction: a drop-phase and an add-phase. CP starts by computing the utility ratio $u_{ij} = v_{ij} / \sum_{k=1}^{m} R^k W_{ij}^k$, $j \in \{1, \dots, r_i\}$, of each item j belonging to each class S_i. Then it selects the item j from each class S_i, $i \in \{1, \dots, n\}$, realizing the most valuable u_{ij}. If the obtained solution is feasible, then CP terminates, otherwise in the drop-phase, it considers the most violated constraint, noted R^{k_0}. It then chooses the class S_{i_0} corresponding to the fixed item j_{i_0} having the highest weight $w_{i_0 j_{i_0}}^{k_0}$ over all the fixed items and regarding the most violated constraint R^{k_0}. The selected item is then swapped with another selected item j from the same class S_{i_0}, and the procedure controls the feasibility (add-phase). If the new obtained solution is not feasible, it selects the lightest item j'_{i_0} from S_{i_0} which in turn is considered as the new selected item. This process is iterated until a feasible (or reduce the amount of unfeasibility) solution is provided (noted F).

As described above, CP provides either a feasible or unfeasible solution to MMKP. For the last case, the unfeasibility of the solution can also be reduced by applying the procedure proposed in [9] which uses a local swapping strategy (the so-called 2-opt strategy) between two items j and j' belonging to the same class S_i. Now in both cases, the column generation procedure uses the columns corresponding to the solution F obtained. At the root node, we also add a column per class i, $i = 1, \dots, n$ (different from the columns provided by F), where each column is taken following the greatest utility ratio, i.e.,

$$\max_{1 \le j \le r_i} \frac{v_{ij}}{\sum_{k=1}^{m} w_{ij}^k}.$$

The last choice means that for each class we favor the column realizing the greatest utility value regarding all knapsack constraints. Note that the strategy currently used remains the best among the strategies than we tested, like including several columns for each class, or including the column realizing the minimal or average utility ratio.

3.4 A search procedure

Branch-and-Bound (B&B) is a well-known technique for solving combinatorial search problems. Its basic scheme is to reduce the problem search space by dynamically pruning unsearched areas which cannot yield better results than already found.

The B&B method searches a finite space T, implicitly given as a set, in order to find one state $t^* \in T$ which is optimal for a given objective function f. Generally, this approach proceeds by developing a tree in which each node represents a part of the state space T. The root node represents the entire state space T. Nodes are branched into new nodes which means that a given part T' of the state space is further split into a number of subsets, the union of which is equal to T'. Hence, the optimal solution over T' is equal to the optimal solution over one of the subsets and the value of the optimal solution over T' is the minimum (or maximum) of the optima over the subsets. The decomposition process is repeated until the optimal solution over the part of the state space is reached.

Herein, we use a three-stage strategy for solving the MMKP. This strategy may be summarized as follows:

1. Starting by an initial solution-lower bound (Sect. 3.3).
2. Generating a subset of nodes depending on the strategies used (Sect. 3.5).
3. Applying a heuristic solution procedure to some selected nodes (Sect. 3.6).

As we shall explained in Sect. 3.6, the heuristic used in Step 3 considers two alternative solutions procedures: (i) the first one considers a greedy algorithm completed with the drop-phase of CP procedure if the provided solution is unfeasible (as discussed in Sect. 3.3) and, (ii) the second one solves, in a greedy way, a part of the current selected node and apply a restricted exact algorithm for the complementary part. Obviously, this approach makes it possible to simulate an exact algorithm but it does not guarantying the optimality of the final solution. For that, we propose thereafter how to incorporate the column generation procedure in order (i) to accelerate the search process and (ii) to provide satisfactory solutions for the MMKP.

3.5 Branching

Generally, using a branch-and-bound procedure, based upon linear programming relaxation, assumes that all fractional solutions can be eliminated by successive separation of the potential feasible solution space. The first natural branching scheme consists in designing a set of rules that enables one to exclude any given fractional solution while making sure the resulting separation of the local program potential feasible solution space is valid. The second branching strategy, in such an approach, takes into consideration the performance of the enumerative approach.

For the MMKP, a maximization problem, if no further variables with positive reduced costs can be found by completely solving the pricing problem, branching becomes necessary.

3.5.1 A first branching

Recall that branching is usually performed by partitioning the search space (for a current selected node/tree) into two parts and by choosing a variable that has a fractional value x_{ij}^f in the LP solution. The two resulting branches are then defined by: $x_{ij} \le \lfloor x_{ij}^f \rfloor$ and $x_{ij} \ge \lceil x_{ij}^f \rceil$. In our study, we followed the common choice by branching on the most fractional variable, i.e. the variable with fractional part closest to one.

3.5.2 A second branching

Regarding the particularity of the MMKP, fixing a variable of a class induces the suppression of the same class. Differently stated, fixing a variable x_{pj} to one induces the fixation of $|S_p| - 1$ variables to zero. By exploiting the particularity of the problem, one can force a fixation of several items to zero for the first branch and, force a single element to one for the second branch. Herein, the first branch corresponds to adding the following constraint associated to the p-th class; that is

$$x_{p1} = x_{p2} = \cdots = x_{p\ell} = 0, \tag{5}$$

and the second branch corresponds to adding the following constraint

$$\sum_{j=\ell+1}^{|S_p|} x_{pj} = 0, \tag{6}$$

where ℓ is the index of the first decision variable having a fractional value. Note that the choice of the first fractional variable preserves the greatest pseudo-utility value. Of course, we also tried other fractional variables like the last fractional value, but limited computational results showed that the strategy currently used realized satisfactory solutions.

3.5.3 A third branching

Let us note that the above branchings are specialized on excluding items in the same class. Herein, we introduce another strategy which can be viewed as a constraint separation for the current node; that is based on separating a node into two complementary new nodes. Indeed, at a node x, let E_x be the set of the last columns added to the node x (recall that at each step of the column generation procedure a column per class is added). Then, the first used branch corresponds to adding the following constraint

$$\sum_{(i,j)\in E_x} x_{ij} \le \frac{|E_x|}{2}, \quad i \in \{1,\dots,n\},\ j \in \{1,\dots,r_i\}, \tag{7}$$

and the second branch corresponds to adding the following constraint

$$\sum_{(i,j)\in E_x} x_{ij} \ge \frac{|E_x|}{2} + 1, \quad i \in \{1,\dots,n\},\ j \in \{1,\dots,r_i\}. \tag{8}$$

Of course, several ways of building the set E_x can derived from the above separation (i.e., injecting constraints of type (7) and (8)). Indeed, one can choose (on the relaxed problem) the fractional elements of the current node x, or a subset of the elements fixed at 1 on this node, or even combine with the elements fixed at 1 and the other fractional elements.

3.5.4 Managing the branching

In this section, we propose a way for managing the branching process. It is clear that in such a process, several strategies can be used. In our study, we tried several ways, but herein we just present the process reaching satisfactory solutions within reasonable average runtime. Of course, as shown in Sect. 4, our solution procedures are tested using the single branching strategy (either the first, the second or the third one is applied) and combining the three strategies described in Sects. 3.5.1, 3.5.2 and 3.5.3.

Let E_x denote the set of the latest columns (variables) added by the column generation procedure at node x, where one column is selected per problem SP_i, $i = 1, \dots, n$. Then,

1. Apply the first variable branching of Sect. 3.5.1, when $|E_x| = 1$.
2. Create a separation by applying the second branching of Sect. 3.5.2 when $|E_x| \geq 2$.
3. Apply the complementary branching of Sect. 3.5.3 if the second branching was applied during the last iteration and the number of the introduced columns is greater than or equal to 2.

3.6 Integer solutions

Each node of the global tree, provided by the restricted exact algorithm, requires the application of the column generation. In addition, the performance of the column generation depends on the computing effort of one iteration when solving the subproblem, as well as the total number of iterations before reaching optimality. Both factors become crucial especially when trying the resolution of large-scale instances within reasonable runtime.

We recall that CGP solves the (restricted) LP-relaxation of MMKP (Sect. 3.2). If some variables are fractional-valued in the LP-optimum, the solution does not represent a feasible solution. To obtain integer solutions, enumeration schemes such as the branch-and-bound or heuristics are necessary. In our study we use two alternative solution procedures. The first alternative solution considers a rounding procedure combined (in some cases) with the drop-phase of CP (Sect. 3.3) and the second one uses a two-phase procedure combining a modified rounding procedure and a truncated exact algorithm.

3.6.1 A rounding solution procedure

A primal integer solution to the original problem is obtained by rounding its LP (approximate) solution step by step in an iterative procedure. The used procedure can be viewed as a two-step approach:

1. At node η, having initialized the master problem as explained above, we optimize its LP relaxation using the column generation procedure. Then, we collect the current primal solution to the restricted master LP. We fix to their current value x_{ij} that are currently integer (with other variables belonging to the same classes) and we select one fractional variable x_{ij} that shall be rounded up. This candidate is chosen as the largest fractional value x_{ij}.

2. The reduced problem that remains after fixing some variables (and classes) (step 1) is submitted to an equivalent treatment. Indeed, its LP relaxation is solved by column generation and the greatest fractional variable x_{ij} is rounded in its LP solution. This process is applied until approximate LP solution of the reduced problem is integer and hence no more rounding is needed. So, a primal integer solution is provided.

Note that at step 2, two cases can be distinguished when the final solution is reached. On the one hand, the primal integer solution is provided and so, it realizes a feasible solution to the original MMKP. On the other hand, the final solution is unfeasible and so, the drop-phase of CP solution is needed for eliminating (or reducing) the amount of unfeasibility of the solution. Of course, for the last case we can also try to fix the selected fractional variable x_{ij} to zero when the unfeasibility of the solution appears.

Moreover, such an approach can provide moderate solutions for the MMKP if the solutions reached is unfeasible. In fact, this phenomenon can be explained by the particularity of the MMKP, since finding a feasible solution to such a problem is also a difficult task. On the other hand, observe that fixing a decision variable to one involves a fixation of a subset of variables to zero (variables corresponding to the same class). Thus, we think that in some cases the rounding procedure may eliminate the classes in a very aggressive way.

3.6.2 An alternative solution procedure

Observe that the pricing phase—solving the problems $SP(i), \forall i \in \{1, \dots, n\}$—requires a negligible time for selecting the columns enriching the subset S_0. We thus propose to transfer the work usually dedicated to the pricing phase to the resolution of a new subproblem. We do it by combining some treatments of the rounding procedure combined with a truncated exact procedure for solving a reduced integer subproblem. Indeed, for each selected node, the used process can be viewed as a two-phase procedure in which both rounding procedure and specialized exact algorithm cooperate for solving the MMKP. These two phases follow:

Phase 1. Let I_1 and I_2 be two subsets such that $|I_1 \cup I_2| = n$ and $I_1 \cap I_2 = \emptyset$, where I_1 denotes the subset containing the already fixed classes (i.e., one of the items associated to each of these classes is fixed to one) and I_2 is the subset of non-fixed (free) classes. Then the following steps are applied:

1. Apply the rounding solution procedure for fixing some classes of the LP solution (construction of both sets I_1 and I_2).
2. Apply the fractional-rounding procedure (on I_2) for fixing $\alpha\%$ of fractional variables (classes) in LP solution, where $\alpha \in]0, 100]$.

Phase 2. Apply a truncated exact algorithm on the rest of the problem and exit with the solution reached.

In order to improve the quality of the solutions obtained, one can apply the above process on a certain number of nodes. Obviously, for the MMKP, it is not easy to decide the choice of the best generated nodes. Herein, we introduce two empirical parameters, namely β_1 and β_2. The parameter β_1 represents the number of generated nodes and β_2 is introduced in order to simulate a diversification on the search space. It means that instead of treating only the first generated nodes (by combining the rounding procedure and a restricted exact procedure), we introduce a periodic resolution which consists in applying a resolution of the selected node among all those generated. The selected nodes can be considered as the elite nodes for which the solution procedure (both rounding solution and hybrid ones) is applied.

Another point, when applying the alternative solution procedure, concerns the stopping strategy when the rest of the problem (the non-fixed variables) is solved using an exact algorithm. In our experimental results, we introduced a simple node-based tree termination scheme for aborting the current tree (elite node). Indeed, each tree will be aborted when their total number of created nodes exceeds a given limit.

4 Computational results

The purpose of this section is twofold: (i) to show how to determine a good trade-off between the solutions obtained and the running time and, (ii) to evaluate the performance of both versions of the algorithm compared to the results reached by the best version of the algorithm proposed in [8], noted MRLS. The obtained results are also compared to those obtained when running one hour the Cplex Solver v.9 on the same set of instances. Our algorithms were coded in C++ and all considered algorithms were tested on an UltraSparc10 (250 Mhz and with 1 Gb of RAM).

The problems used as benchmarks are summarized in Table 1. We tested a total of 33 instances corresponding to two groups. The first group contains 13 instances (noted I01, ..., I13) varying from small to large-scale size ones. These instances are already used by Khan et al. [12]. The second group, taken from [8], is composed of 20 problem instances (noted Ins01, ..., Ins20) varying from medium to large ones.

4.1 Behavior of the rounding procedure

Generally, when using approximate algorithms to solve optimization problems, it is well-known that different parameter settings for the approach lead to results of variable quality. Our *rounding procedure* (noted RP) and the alternative *column generation based algorithm* (noted CGBA) involve several parameters. Indeed, both algorithms involve the parameter associated to the number of the trees (elite nodes) to solve and the frequency parameter used for extending the search to other regions. CGBA involves also the percentage of items to fix to their integer values (in both sets I_1 and I_2). Of course, a different adjustment of method's parameters would lead a high percentage of good solutions. But this better adjustment would sometimes lead to heavier execution time requirements. The set of values chosen in our experiment represents a satisfactory trade-off between the solutions and the running time.

First, in order to find the right value of the maximum number of solved trees by the algorithm, we introduced a variation of β_1 in the discrete interval $\{200, 500, 1000\}$. These tests were made by fixing the frequency parameter β_2 in the discrete interval $\{5, 10, 25\}$, representing the number of branchings used before developing the last

Table 1 Test problem details: The first group contains instances varying from small to large ones and the second one is composed of medium and large-scale ones

Group 1					Group 2				
#Inst.	n	r_i	m	$\sum_{i=1}^{n} r_i$	#Inst.	n	r_i	m	$\sum_{i=1}^{n} r_i$
I01	5	5	5	25	Ins01	50	10	10	500
I02	10	5	5	50	Ins02	50	10	10	500
I03	15	10	10	150	Ins03	60	10	10	600
I04	20	10	10	200	Ins04	70	10	10	700
I05	25	10	10	250	Ins05	75	10	10	750
I06	30	10	10	300	Ins06	75	10	10	750
I07	100	10	10	1000	Ins07	80	10	10	800
I08	150	10	10	1500	Ins08	80	10	10	800
I09	200	10	10	2000	Ins09	80	10	10	800
I10	250	10	10	2500	Ins10	90	10	10	900
I11	300	10	10	3000	Ins11	90	10	10	900
I12	350	10	10	3500	Ins12	100	10	10	1000
I13	400	10	10	4000	Ins13	100	30	10	3000
					Ins14	150	30	10	4500
					Ins15	180	30	10	5400
					Ins16	200	30	10	6000
					Ins17	250	30	10	7500
					Ins18	280	20	10	5600
					Ins19	300	20	10	6000
					Ins20	350	20	10	7000

generated elite node (tree)—other values have been considered, but we only report the significant ones. Note that generating β_1 nodes means that $\beta_1/2$ nodes can be treated by the rounding procedure because each branching strategy creates two new potential nodes.

Table 2 displays the results provided by RP when applying/combining the different branching strategies. For each line-bloc, the first column tallies the branching strategy used, the average runtime that needs each version of RP and the number of the best solutions provided when both parameters β_1 and β_2 vary in the discrete interval $\{200, 500, 1000\}$, and $\{5, 10, 25\}$, respectively. Note that, for the first six instances, all versions of RP produced the optimal solutions.

As shown in Table 2, we remark that:

- RP with $\beta_2 = 25$ provides moderate results even if the average runtime remains rather small. Indeed, in this case RP is able to reach between 1 and 8 best solutions and by consuming an average runtime varying from 5.78 to 29.94 seconds.
- RP has a better behavior for $\beta_2 \in \{5, 10\}$ and when varying β_1 in $\{500, 1000\}$. In this case, the number of the best solutions produced varies from 4 to 17 (out of 27 instances) and the average runtime varies from 22.57 to 126.07 seconds.

Table 2 The behavior of RP when varying the number of nodes, the frequency-parameter and the branching strategy

	β_1	$\beta_2 = 25$		$\beta_2 = 10$		$\beta_2 = 5$	
		Av. CPU	# Best	Av. CPU	# Best	Av. CPU	# Best
Branching 1							
	200	7.10	1	18.87	3	27.34	5
	500	13.71	3	33.17	4	49.59	8
	1000	22.56	5	52.40	10	107.13	12
Branching 2							
	200	5.78	2	14.50	2	20.89	5
	500	9.10	3	22.57	6	38.82	10
	1000	20.13	6	52.22	9	100.17	12
Branching 3							
	200	9.49	2	22.03	5	30.33	5
	500	14.46	4	37.97	8	52.53	8
	1000	27.20	5	69.47	10	106.72	10
Branchings 1 and 2							
	200	9.23	2	20.44	6	25.52	6
	500	14.53	4	32.16	7	40.75	7
	1000	20.48	5	50.01	13	104.93	13
Branchings 1 and 3							
	200	9.99	2	25.39	4	30.92	4
	500	14.93	3	37.87	7	43.85	7
	1000	21.29	3	57.94	10	110.44	10
Branchings 2 and 3							
	200	7.70	2	18.24	7	24.03	7
	500	9.46	4	24.09	12	43.89	12
	1000	19.56	7	52.31	13	109.56	13
Branchings 1, 2 and 3							
	200	7.59	2	14.23	3	23.63	7
	500	11.62	5	22.79	15	45.18	15
	1000	29.94	8	61.71	15	126.07	17

- Regarding the number of better solutions provided, RP with the last strategy (which combines the three branching strategies) can be considered as a better one. Indeed, on the one hand, for each column-bloc characterizing the variation of β_1, the number of best solutions increases within globally the same average runtime. On the other hand, from the last line-bloc, the maximum number of better-solutions is obtained when fixing the parameter β_1 to 1000, but it needs a largest average runtime (last line, column 7).
- In addition, one can notice that the variation of the ratio $\frac{\text{Av.CPU}}{\text{\#Best}}$ is more advantageous for the adjustment $\beta_1 = 500$ and $\beta_2 = 10$ (it realizes an average value of

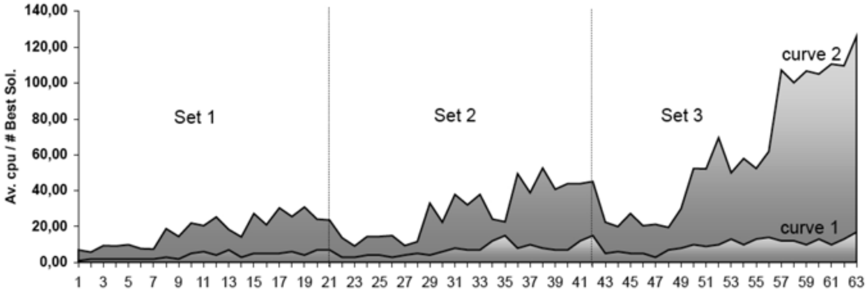

Fig. 1 Variation of the average runtime versus the number of the provided best solutions. For the first six instances, all versions of RP provide the optimal solutions

1.52 compared to the average value of 7.42 representing the adjustment $\beta_1 = 1000$ and $\beta_2 = 25$).

Figure 1 shows the behavior of RP when varying the branching strategies and both parameters β_1 and β_2. Three sets are represented: a first set corresponding to $\beta_1 = 200$, noted Set 1, a second set (noted Set 2) corresponding to $\beta_1 = 500$ and a third one for $\beta_1 = 1000$, noted Set 3. Each of these sets (Set 1, Set 2 and Set 3) contains the results of the three bloc-columns of Table 2 successively taken from the first bloc-column (i.e. $\beta_2 = 25$) to the third bloc-column (i.e. $\beta_2 = 5$). From Fig. 1, we observe that:

- First, for $\beta_1 = 200$, RP is fast (see curve 2, Set 1) but it reaches a small number of best solutions (see curve 1, Set 1). In this case, the highest value 7, representing the best solutions reached, corresponds to the values $\beta_2 = 5$ and 10, which use the branching strategies 2 and 3, and 1, 2 and 3, respectively.
- Second, for $\beta_1 = 500$ and according to the curve 1, we can distinguish two interesting tunings which correspond to the value 15: (i) the first tuning corresponds to the value 35 on the X axis, which also represents the couple (Av. CPU, #Best) = (22.79, 15), and (ii) the second one, at the position 42, corresponds to the couple (Av. CPU, #Best) = (45.18, 15). Both values corresponds to $\beta_2 = 10$ and 5, respectively.
- Third, in some cases RP is able to reach 17 better solutions (see curve 1, Set 3), but it needs more computational time (see curve 2, Set 3).

We can conclude that a high value of the parameter β_1 does not necessary reach the best solutions and the runtimes considerably increase. In what follows, we maintain the values corresponding to the couple $(\beta_1, \beta_2) = (500, 10)$ that allowed to realize a good trade-off between the number of the best solutions obtained and the average runtime.

Table 3 evaluates the performance of RP; its results are compared to the results reached by both Cplex solver and MRLS. Column 2 displays the best solution (or the optimal solution; in this case, the instance is marked with a "$\circ$" sign) of the instance. Column 3 contains the best *Integer Feasible Solution Cplex*$_{\text{IFS}}$ provided by the Cplex solver. Column 4 (resp. column 5) tallies the solutions (resp. runtime) given by (resp. that needs) MRLS. Finally, column 6 reports the quality of the obtained solution

Table 3 Performance of the rounding procedure. The symbol $\star$ means that the algorithm uses the drop-phase of CP procedure for providing a feasible solution

Inst	Best/Opt Sol	Cplex solver	MRLS		RP	
		$\text{Cplex}_{\text{IFS}}$	Sol.	CPU	Sol.	CPU
I01	173°	173	173	0.59	173	0.16
I02	364°	364	364	0.81	364	0.73
I03	1602°	1602	1602	2.01	1602	1.19
I04	3597°	3597	3597	2.30	3597	2.21
I05	3905.7°	3905.7	3905.7	1.94	3905.7	0.05
I06	4799.3°	4799.3	4799.3	2.37	4799.3	1.55
I07	24587	24584	24587	36.58	24587	11.17
I08	36877	36869	36877	37.00	36869	19.80
I09	49167	49155	49167	25.10	*49175*	18.92
I10	61446	61446	61437	47.00	*61461*	28.87
I11	73773	73759	73773	41.45	73759	40.59
I12	86071	86071	86069	42.08	86071	48.07
I13	98429	98418	98429	160.41	98409	58.85
Ins01	10714	10709	17014	10.27	*10719*	6.21
Ins02	13598	13597	13598	76.00	13460	4.76
Ins03	10943	10934	10943	58.00	10939*	4.97
Ins04	14429	14422	14429	7.69	*14436*	8.15
Ins05	17053	17041	17053	42.00	17047*	9.70
Ins06	16823	16815	16823	50.00	16823	8.87
Ins07	16423	16407	16423	65.00	16310	11.42
Ins08	17506	17484	17506	26.78	*17510*	9.57
Ins09	17754	17747	17754	51.23	*17760*	8.94
Ins10	19314	19285	19314	32.16	19306*	11.43
Ins11	19431	19424	19431	110.98	19431	9.53
Ins12	21730	21725	21730	23.39	21646	14.32
Ins13	21569	21569	21569	18.00	21550	18.89
Ins14	32869	32866	32869	72.00	*32870*	25.79
Ins15	39154	39154	39148	63.00	*39157*	37.53
Ins16	43357	43357	43354	194.00	*43361*	40.69
Ins17	54349	54349	54349	30.00	54329	55.67
Ins18	60456	60455	60456	201.00	*60457*	65.77
Ins19	64921	64919	64921	45.00	64913	74.72
Ins20	75603	75603	75603	47.00	*75610*	92.99
Av. CPU				49.19		22.79

whereas column 7 displays the computational time that needs RP for reaching the final solution. All entries in italic (Column 6) indicate if RP improves the best solution (reached by either Cplex solver or MRLS). The analyze of the results of Table 3 follows.

First, for the first group of instances, we can remark that RP matches the solutions corresponding to the first six instances whose optimal solutions are known (for these instances, all algorithms produce the optimal solutions), it matches one (resp. two) solution(s) reached by MRLS (resp. Cplex), produces 2 new better solutions out of 6 and fails in 3 (resp. one) occasions to reach the solutions produced by MRLS (resp. Cplex).

Second, for the second group of instances, RP reaches 9 new better solutions out of the 20 treated instances, it matches 2 solutions reached by MRLS and fails in 9 (resp. 6) occasions to reach the solutions produced by MRLS (resp. Cplex). Note also that RP fails in 3 cases out of 20 to reach a feasible solution; in this case, as we have mentioned in Sect. 3.6.1, we attempt to construct a feasible solution by applying the complementary phase (the drop-phase of CP solution) to the last two nodes.

Third and last, MRLS (resp. Cplex) needs 49.19 seconds (resp. one hour) whereas RP provides the results displayed in Table 3 by consuming 22.79 seconds, on average.

4.2 Performance of CGBA

In this section, we compare the results produced by CGBA to those provided by MRLS and Cplex solver on both groups of instances. This comparison is performed by setting the runtime limit of the Cplex solver to one hour. The results corresponding to MRLS are taken from Hifi et al. [8] and CGBA is tested by considering different adjustments. Herein, we maintain the values of β_1 and β_2 used by RP (see Sect. 4.1). We recall that, on the one hand, CGBA considers two subsets I_1 and I_2 (the first subset I_1 contains the elements whose values are integers, the second subset I_2 contains the rest of the free items), and on the other hand, other supplementary parameters and strategies are used: (i) α_1 (the number of the global percentage of classes to fix to their integer variables), (ii) α_2 (the percentage of items to fix to their integer values in I_1), and (iii) the branching strategy used by the algorithm. Note that, on the one hand, both parameters α_1 and α_2 are introduced in order to restrict the percentage of variables to fix to their integers values. On the other hand, the percentage of the fractional variables to be round-up, denoted α, is replaced by the most fractional value.

First, we analyze the behavior of CGBA when varying both parameters α_1 and α_2 used by CGBA. The results displayed in Table 4 are provided by using the branching strategy described in Sect. 3.5.4 (we shall discuss below the choice of the branching strategies). Table 4 reports the number of the best solutions provided (attained) when both parameters α_1 and α_2 vary in the discrete interval $\{25\%, 50\%, 75\%\}$, and the average runtime of each version of the algorithm. The first line-bloc of the table displays the results when fixing α_2 to 75%, the second line-bloc tallies the provided results for $\alpha_2 = 50\%$ and the third one concerns the results when setting α_2 to 25%. Finally, note also that in some cases CGBA needs to limit the number of nodes generated when the Cplex solver is applied on the rest of the free variables. Herein, in order to maintain the same degree of comparison, in particular for the average runtime, we considered a maximum of 5000, 4000 and 3000 nodes per tree for $\alpha_1 = 25\%, 50\%$ and 75%, respectively. From Table 4, we observe that:

Table 4 The behavior of CGBA when varying the parameter α_1

	$\alpha_1 = 75\%$		$\alpha_1 = 50\%$		$\alpha_1 = 25\%$	
	Av. CPU	Nb. Best/Opt	Av. CPU	Nb. Best/Opt	Av. CPU	Nb. Best/Opt
$\alpha_2 = 75\%$						
First group	72,03	7	83,00	7	87,87	10
Second group	105.92	8	130.14	11	156.96	12
Both groups	92.57	15	111.57	18	129.74	22
$\alpha_2 = 50\%$						
First group	79.88	9	93.40	12	111.34	12
Second group	114.45	10	188.43	18	240.96	17
Both groups	100.83	19	150.99	30	189.90	29
$\alpha_2 = 25\%$						
First group	107.98	8	125.52	11	150.83	12
Second group	138.68	11	179.90	17	280.12	16
Both groups	126.59	19	158.48	28	229.19	28

- For $\alpha_1 = \alpha_2 = 75\%$, CGBA reaches 15 better solutions out of 33 and it can be considered as a faster version. In this case, it consumes an average runtime of 92.57 seconds.
- There are four versions of the algorithm which provide closely the same number of the best solutions (between 28 and 30 better solutions). Indeed, four couples of (α_1, α_2)—corresponding to $(50\%, 50\%)$, $(50\%, 25\%)$, $(25\%, 50\%)$ and $(25\%, 25\%)$—realize a number of $30,\ 29,\ 28$ and 28 best solutions, respectively.
- Among the four versions reaching the solutions, we distinguish two faster versions. Indeed, the first version corresponding to the couple $(50\%, 50\%)$ needs an average runtime of 150.99 seconds, and the second one, characterizing the couple $(25\%, 50\%)$, needs a slightly more important average runtime (i.e., 158.48 seconds).

From Table 4, we can conclude that the intermediate values for the couple (α_1, α_2) maintain the highest percentage of the best solutions reached within reasonable average runtime.

Second, in order to analyze the behavior of CGBA when using the branching strategies, we considered seven versions of the algorithm. Three versions of CGBA use a single branching strategy (either the first, the second or the third one), three other versions combining two different branching strategies and the last one which uses the alternative branching strategy (as described in Sect. 3.5.4). In order to make a more complete comparison between the seven versions of the algorithm, we fix α_2 to 50% (regarding the good behavior of the algorithm with this value—see the second line-bloc of Table 4) and we vary the value of α_1 in the interval $\{25\%, 50\%, 75\%\}$. The obtained results, when using the different tuning, are reported in Table 5.

A thorough discussion of the results (for both groups of instances), displayed in Table 5, follows:

Table 5 The behavior of CGBA using the branching strategies

	$\alpha_1 = 75\%$		$\alpha_1 = 50\%$		$\alpha_1 = 25\%$	
	Av. CPU	Nb. Best/Opt	Av. CPU	Nb. Best/Opt	Av. CPU	Nb. Best/Opt
Branching 1						
First group	59.26	7	72.31	8	90.07	10
Second group	187.40	7	229.85	14	249.56	15
Both groups	136.92	14	167.79	22	186.73	25
Branching 2						
First group	50.06	8	58.91	10	69.75	11
Second group	89.31	9	117.44	14	162.52	16
Both groups	73.85	17	94.38	24	125.97	27
Branching 3						
First group	101.66	8	138.49	9	159.32	9
Second group	181.44	7	232.74	12	300.67	16
Both groups	150.01	15	195.61	21	244.98	25
Branchings 1 and 2						
First group	89.43	8	101.31	11	144.18	10
Second group	140.51	10	186.09	13	283.09	16
Both groups	120.39	18	151.65	24	226.66	26
Branchings 1 and 3						
First group	79.84	9	997.29	12	139.11	11
Second group	185.34	8	228.61	10	274.72	14
Both groups	143.78	17	176.88	22	221.30	25
Branchings 2 and 3						
First group	90.20	9	121.56	10	156.82	12
Second group	148.37	10	197.44	15	279.55	15
Both groups	125.46	19	167.55	25	231.20	27
Branchings 1, 2 and 3						
First group	79.88	9	93.40	12	111.34	12
Second group	114.45	10	188.43	18	240.96	17
Both groups	100.83	19	150.99	30	189.90	29

- First, globally, the percentage of the best solutions produced by CGBA varies from 42.42% to 90.91% whereas the average runtime varies from 73.85 to 244.98 seconds.
- Second, CGBA with the first branching strategy has a good behavior when α_1 is fixed to 25%. Indeed, in this case it reaches 25 out of 33 best solutions and it requires a rather important average runtime, i.e. 186.73 seconds.
- Third, the analysis of the second branching strategy is very interesting, as shown from the second line-bloc. First, the algorithm with $\alpha_1 = 75\%$ is able to match 17 better/optimal solutions by consuming only 73.85 seconds, in average. Second, for $\alpha_1 = 50\%$, the algorithm increases the number of the best solutions provided (it

becomes now equal to 24) by consuming a slightly more important average runtime (i.e. 94.38 seconds). Third and last, CGBA with $\alpha_1 = 25\%$ is able to maintain an interesting average runtime (i.e. 125.97 seconds) and to increase the number of the best solutions reached with 3 units.

- Fourth, as shown from the third line-bloc, the algorithm with the third branching strategy has an equivalent behavior compared to the results reached when the first branching strategy is used.
- Fifth, regarding the first results provided by the algorithm, we mainly limit the analysis on the combination of the second branching strategy with the other ones, i.e. the first or the third one. First, as shown from the fourth (resp. sixth) line-bloc, we observe that combining the first (resp. third) branching strategy with the second one becomes benefits for the approach regarding the number of the provided best/optimal solutions. However, in some cases the average runtime increases considerably: (i) it varies from 120.39 to 226.66 seconds when combining the first two branching strategies, and from 125.46 to 231.20 when the last two branching strategies are combined.
- Sixth and last, the behavior of the algorithm is more interesting when combining all branching strategies following the scheme described in Sect. 3.5.4. As observed from the last line-bloc, first, CGBA with $\alpha_1 = 75\%$ maintains the high number of better/optimal solutions of the first column. Second, CGBA with $\alpha_1 = 50\%$ produces a highest number of best/opt solutions by consuming 150.99 seconds, on average. Third and last, for $\alpha_1 = 25\%$, it maintains the high percentage of the best/optimal solutions produced but it needs more average runtime (i.e., 189.90 seconds).

Hence, globally, the average runtime of CGBA with the three branching strategies can be considered as an interesting one regarding the good quality of the provided results.

Table 6 reports, for both groups, the detailed results of CGBA when (i) only the second branching strategy is applied with $\alpha_1 = 25$, and (ii) the three branching strategies are combined by varying α_1 in $\{25, 50, 75\}$. We also report, for the same groups, the solutions of MRLS and the Cplex solver. Column 2 shows the best solution (or the optimal solution; in this case, the instance is marked with a "$\circ$" sign) of each instance produced by one of the considered algorithms. Column 3 (resp. column 4) contains the solution reached by the Cplex solver (resp. MRLS). Column 5 tallies the solutions provided by RP. Columns 6, 8, 10 and 12 tally the solutions given by the four versions of CGBA whereas columns 7, 9, 11 and 13 display the runtime that needs each version of CGBA. All entries with the symbol "–" indicate which algorithm reaches the best solution for the considered instance.

The analysis of the results of Table 6 follows:

- Globally, either implementation of CGBA reaches better solutions than Cplex solver, MRLS and RP.
- If we consider the reference solution as the best solution produced by the Cplex solver, MRLS and RP, then the fast version of CGBA—using all branching strategies—with $\alpha_1 = 75\%$ (Columns 8 and 9) produces 8 better solutions out of 33, it matches 22 solutions and it fails in 3 occasions to reach the reference solution.

Table 6 Performance of CGBA on two groups of instances. The symbol "–" means that the algorithm provides the best solution

					CGBA: column generation based algorithm							
		Cplex solver	MRLS	RP	Branching 2, $\alpha_1 = 25\%$		Branchings 1, 2 and 3					
							$\alpha_1 = 75\%$		$\alpha_1 = 50\%$		$\alpha_1 = 25\%$	
Inst	Best/Opt	$\text{Cplex}_{\text{IFS}}$	Sol.	Sol.	Sol.	CPU	Sol.	CPU	Sol.	CPU	Sol.	CPU
I01	173	–	–	–	–	0.28	–	0.21	–	0.23	–	0.27
I02	364	–	–	–	–	1.04	–	1.87	–	2.33	–	2.31
I03	1602	–	–	–	–	5.67	–	3.48	–	3.52	–	4.58
I04	3597	–	–	–	–	10.69	–	6.42	–	9.78	–	13.81
I05	3905.70	–	–	–	–	0.09	–	0.07	–	0.08	–	0.08
I06	4799.30	–	–	–	–	0.96	–	4.79	–	4.36	–	4.38
I07	24587	24584	–	–	–	95.83	24584	34.09	–	63.95	–	74.71
I08	36892	36869	36877	36869	36888	138.5	36887	94.35	36890	127.08	–	170.06
I09	49176	49155	49167	49175	–	72.18	49175	72.93	–	96.20	49175	108.24
I10	61461	61446	61437	–	–	98.61	–	119.38	–	151.49	–	188.73
I11	73775	73759	73773	73759	–	103.92	–	146.16	–	169.72	–	194.35
I12	86078	86071	86069	86071	–	211.63	86071	219.96	–	229.26	–	264.95
I13	98431	98418	98429	98409	98429	167.32	–	334.72	–	356.24	–	421.01
Ins01	10732	10709	17014	10719	–	15.65	10719	16.06	–	46.26	10730	57.14
Ins02	13598	13597	–	13460	13597	22.16	–	15.50	–	12.78	–	20.85
Ins03	10943	10934	–	–	–	80.57	–	11.60	–	48.37	–	58.36
Ins04	14440	14422	14429	14436	–	85.96	14438	27.53	–	62.53	–	66.38
Ins05	17053	17041	–	–	–	87.32	–	55.92	–	91.16	–	119.86
Ins06	16825	16815	16823	16823	–	35.91	16823	34.18	16824	75.62	–	108.88
Ins07	16435	16407	16423	16310	16432	76.5	16424	39.69	–	122.40	16432	126.17

Table 6 (*Continued*)

Inst	Best/Opt	Cplex solver	MRLS	RP	CGBA: column generation based algorithm							
					Branching 2, $\alpha_1 = 25\%$		Branchings 1, 2 and 3					
							$\alpha_1 = 75\%$		$\alpha_1 = 50\%$		$\alpha_1 = 25\%$	
		$\text{Cplex}_{\text{IFS}}$	Sol.	Sol.	Sol.	CPU	Sol.	CPU	Sol.	CPU	Sol.	CPU
Ins08	17510	17484	17506	–	–	37.21	–	36.33	–	74.68	–	83.08
Ins09	17760	17747	17754	–	–	112.56	–	29.03	–	48.23	–	56.11
Ins10	19314	19285	–	19306	–	65.84	19312	68.13	–	143.45	–	188.64
Ins11	19434	19424	19431	19431	–	46.25	19432	33.49	–	56.55	–	170.48
Ins12	21731	21725	21730	21646	21730	229.97	21728	157.62	–	126.58	21730	227.91
Ins13	21575	21569	21569	21550	21571	89.65	21570	153.25	–	198.53	–	184.95
Ins14	32870	32866	32869	–	–	71.16	–	81.75	–	151.51	–	197.28
Ins15	39157	39154	39148	–	–	135.62	–	154.00	–	306.27	–	397.94
Ins16	43361	43357	43354	–	–	320.19	–	162.39	–	263.90	–	328.43
Ins17	54349	–	–	54329	–	386.72	–	212.15	–	269.07	–	313.53
Ins18	60460	60455	60456	60457	–	227.33	60457	309.93	60459	654.59	–	907.84
Ins19	64923	64919	64921	64913	–	450.58	64921	342.33	–	461.14	–	532.47
Ins20	75611	75603	75603	75610	–	673.25	–	348.18	–	554.88	–	672.98
Av. CPU						125.97		100.83		150.99		189.90
Nb Best		7	12	15	27		19		30		29	

- By using the reference solution, we can observe that CGBA with the second branching strategy remains competitive. Indeed, it is able to provide 13 better solutions by consuming 125.97 seconds, on average. It matches 19 solutions and fails in one occasion to reach the reference solution. Note that this version can be considered as an interesting intermediate solution procedure for the MMKP.
- The last two versions of CGBA provide better solutions (Columns 10–13). Indeed, the version with $\alpha_1 = 25\%$ (resp. $\alpha_1 = 50\%$) is able to reaches 13 (resp. 15) better solutions and to match 20 (resp. 18) solutions.
- Note also that with $\alpha_1 = 25\%$ (resp. $\alpha_1 = 50\%$), CGBA is able to improve 20 (resp. 21) out of 27 solutions, compared to the results reached by both MRLS and Cplex solver. Evidently, these improvements occur at the cost of a larger runtime compared to the runtime of MRLS. However, the improvement of the solution quality warrants the additional (reasonable) runtime.

5 Conclusion

We solved the multiple-choice multi-dimensional knapsack problem using a column generation procedure based algorithm. We proposed two solution procedures: (i) a fast greedy solution procedure and (ii) an alternative one. The first approach is based upon the greedy rounding procedure. The second approach combines two complementary procedures: (i) a rounding solution which is used for fixing a first part of variables of the current problem and (ii) a restricted exact solution procedure which tries to construct better solutions. Computational results show that the first solution procedure is able to provide good solutions within a very short runtime. For the same instances, most of which cannot be solved to proven optimality in a reasonable time, the second approach yields high quality solutions, reaching the optimal/best for several instances, within a reasonable runtime.

Acknowledgements The authors thank the anonymous referees for their helpful comments and suggestions which contributed to the improvement of the presentation and the contents of this paper.

References

1. Akbar, M.M., Sadid, M.W.H., Rahman, M.S., Newton, M.A.H., Kaykobad, M.: A Parallel heuristic algorithm for multiple-choice multi-dimensional knapsack problem. Working paper, Department of CSE, BUET, Dhaka-1000, Bangladesh (2005)
2. Balas, E., Zemel, E.: An algorithm for large zero-one knapsack problem. Oper. Res. **28**, 1130–1154 (1980)
3. Björklund, P., Värbrand, P., Yuan, D.: A column generation method for spatial TDMA scheduling in ad hoc networks. Working paper, Department of Science and Technology, Linköping Institute of Technology, Norrköping, Sweden. To appear in Ad Hoc Netw.
4. Fayard, D., Plateau, G.: An algorithm for the solution of the 0-1 knapsack problem. Computing **28**, 269–287 (1982)
5. Garey, M.R., Johnson, D.S.: Computers and Intractability. Freeman, New York (1979)
6. Gilmore, P.C., Gomory, R.E.: A linear programming approach to the cutting stock problem. Oper. Res. **9**, 849–859 (1961)
7. Gilmore, P.C., Gomory, R.E.: A linear programming approach to the cutting stock problem, part II. Oper. Res. **14**, 94–120 (1963)

8. Hifi, M., Michrafy, M., Sbihi, A.: A reactive local search-based algorithm for the multiple-choice multi-dimensional knapsack problem. Comput. Optim. Appl. **33**, 271–285 (2006)
9. Hifi, M., Michrafy, M., Sbihi, A.: Heuristic algorithms for the multiple-choice multidimensional knapsack problem. J. Oper. Res. Soc. **55**, 1323–1332 (2004)
10. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer, Berlin (2005). ISBN 3-540-40286-1
11. Khan, S.: Quality adaptation in a multi-session adaptive multimedia system: model and architecture. Ph.D. thesis, Department of Electronics and Computer Engineering, University of Victoria (1998)
12. Khan, S., Li, K.F., Manning, E.G., Akbar, MD.M.: Solving the knapsack problem for adaptive multimedia systems. Int. J. Studia Inf. **2**, 154–174 (2002). Special Issue on Cutting, Packing and Knapsacking Problems
13. Mehrotra, A., Trick, M.A.: A column generation approach for graph coloring. INFORMS J. Comput. **8**, 344–354 (1996)
14. Moser, M., Jokanović, D.P., Shiratori, N.: An algorithm for the multidimensional multiple-choice knapsack problem. IEECE Trans. Fundam. Electron. **80**, 582–589 (1997)
15. Toyoda, Y.: A simplified algorithm for obtaining approximate solution to zero-one programming problems. Manag. Sci. **21**, 1417–1427 (1975)
16. Wolsey, L.A.: Integer Programming. Wiley-Interscience, New York (1998)