CORAL: An Exact Algorithm for the Multidimensional Knapsack Problem

Renata Mansini, M. Grazia Speranza,

INFORMS Journal on Computing
Vol. 24, No. 3, Summer 2012, pp. 399–415
ISSN 1091-9856 (print) | ISSN 1526-5528 (online)

http://dx.doi.org/10.1287/ijoc.1110.0460

CORAL: An Exact Algorithm for the Multidimensional Knapsack Problem

Renata Mansini
Department of Information Engineering, University of Brescia, 25123 Brescia, Italy, rmansini@ing.unibs.it

M. Grazia Speranza
Department of Quantitative Methods, University of Brescia, 25122 Brescia, Italy, speranza@eco.unibs.it

The multidimensional knapsack problem (MKP) is a well-known, strongly NP-hard problem and one of the most challenging problems in the class of the knapsack problems. In the last few years, it has been a favorite playground for metaheuristics, but very few contributions have appeared on exact methods. In this paper we introduce an exact approach based on the optimal solution of subproblems limited to a subset of variables. Each subproblem is faced through a recursive variable-fixing process that continues until the number of variables decreases below a given threshold (*restricted core* problem). The solution space of the restricted core problem is split into subspaces, each containing solutions of a given cardinality. Each subspace is then explored with a branch-and-bound algorithm. Pruning conditions are introduced to improve the efficiency of the branch-and-bound routine. In all the tested instances, the proposed method was shown to be, on average, more efficient than the recent branch-and-bound method proposed by Vimont et al. [Vimont, Y., S. Boussier, M. Vasquez. 2008. Reduced costs propagation in an efficient implicit enumeration for the 0-1 multidimensional knapsack problem. *J. Combin. Optim.* **15**(2) 165–178] and CPLEX 10. We were able to improve the best-known solutions for some of the largest and most difficult instances of the OR-LIBRARY data set [Chu, P. C., J. E. Beasley. 1998. A genetic algorithm for the multidimensional knapsack problem. *J. Heuristics* **4**(1) 63–86].

Key words: multidimensional knapsack problem; exact algorithm; reduced costs; recursive variable fixing; cardinality constraint
History: Accepted by Karen Aardal, Area Editor for Design and Analysis of Algorithms; received July 2009; revised July 2010, January 2011; accepted March 2011. Published online in *Articles in Advance* June 17, 2011.

1. Introduction

Let us consider a set $N = \{1, 2, \ldots, n\}$ of items with profits $p_j > 0$, $j = 1, \ldots, n$ and a set $M = \{1, 2, \ldots, m\}$ of resources with availability $c_i > 0$, $i = 1, \ldots, m$. Each item j requires an amount $w_{ij} \geq 0$ of each resource i. The amount w_{ij} can be 0 for some i, j, as long as $\sum_{i=1}^{m} w_{ij} \geq 1$ holds for all items $j = 1, \ldots, n$. Moreover, $w_{ij} \leq c_i$, $j = 1, \ldots, n$, $i = 1, \ldots, m$ and $\sum_{j=1}^{n} w_{ij} \geq c_i$, $i = 1, \ldots, m$. It is assumed that all p_j, w_{ij}, and c_i are integer values. The multidimensional knapsack problem (MKP) can be stated as the problem that looks for a subset of items whose sum of profits is maximum without exceeding resources availability:

$$\text{(MKP)} \qquad \max \sum_{j=1}^{n} p_j x_j \tag{1}$$

$$\sum_{j=1}^{n} w_{ij} x_j \leq c_i \quad i \in M, \tag{2}$$

$$x_j \in \{0, 1\} \quad j \in N. \tag{3}$$

The MKP is known to be one of the most difficult problems in the class of knapsack problems. The MKP is not treated in the famous book by Martello and Toth (1990). A recent and complete survey on the MKP is owed to Kellerer et al. (2004) and includes a large bibliography. The MKP finds application in different practical domains (see, for instance, Mansini and Speranza 2002a). Several heuristics have been proposed in the literature to solve the problem. Some of them are specialized heuristics and metaheuristics (see Chu and Beasley 1998, Glover and Kochenberger 1996, Vasquez and Hao 2001, Vasquez and Vimont 2005, Puchinger et al. 2010), whereas others are more general-purpose methods (see Hanafi and Wilbaut 2011, Wilbaut and Hanafi 2009, Angelelli et al. 2010). All these algorithms can be easily compared because they were tested on the OR-LIBRARY data set introduced by Chu and Beasley (1998).

Very few contributions are available on exact approaches. Shih (1979) designed the first linear programming-based branch-and-bound method for the MKP and reported results on randomly generated uncorrelated instances with 30–90 variables and five constraints. A branch-and-bound algorithm proposed by Gavish and Pirkul (1985) solved problems with sizes up to 80 variables and seven constraints. Larger instances were solved in the bidimensional case (Martello and Toth 2003). More recently, Vimont et al. (2008) proposed an implicit enumeration for the MKP based on the idea that the unpromising parts of the search tree should be tackled first. In their method, the nonbasic variables in the continuous relaxation are selected for the branching and are set first to the complement of their relaxed value. The authors' objective was to prune the search tree as soon as possible. From an experimental point of view, they obtained new optimal solutions on OR-LIBRARY instances with 250 items and provided tighter bounds on the number of items at the optimum on harder instances (with 500 items). The implicit enumeration they proposed is characterized by two key features. The first one is the use of a constraint both at the local and global levels to set nonbasic variables to their optimal values. Such a constraint expresses the objective function of the continuous relaxation through the reduced costs coefficients and bounds this expression with an available lower bound and the optimal objective value. The second feature is related to the idea that the number of items in the optimal solution can be easily bounded between a minimum $k_{\min}$ and a maximum $k_{\max}$ value so that the problem can be partitioned into $(k_{\max} - k_{\min} + 1)$ subproblems, each one exploring solutions of a given cardinality, as proposed in Vasquez and Hao (2001). Finally, Boussier et al. (2008) proposed a new exact method that hybridizes the resolution search by Chvátal (1997) and a branch-and-bound algorithm inspired by the work from Vimont et al. (2008). They solved to optimality all instances with 500 items and 10 constraints requiring, however, an average time of several days.

In this paper we introduce an exact approach called CORAL (for CORe ALgorithm) that makes use of the same two key features used by Vimont et al. (2008) but in a different and more effective way. The idea of tackling the problem by solving a sequence of subproblems, each with a different cardinality constraint, has also been used by Vasquez and Hao (2001) and by Mansini and Speranza (2002b) in an exact approach for the MKP used to model the winner determination problem in multi-unit combinatorial auctions. The use of a reduced-cost constraint based on the objective function of the continuous relaxation has also been applied to other knapsack problems (see Fayard and Plateau 1982 and, more recently, Oliva et al. 2001, where a branch-and-bound approach is proposed).

CORAL can be seen as a continuous lower bound improvement procedure that works through the optimal solution of subproblems limited to subsets of variables. Each subproblem is optimally solved through a two-step procedure. In the first step, the set of items is reduced through a recursive variable-fixing process up to a predefined size. Then, in the second step, the remaining subproblem (restricted core problem) is optimally solved. The solution space is split into subspaces, each containing solutions of a given cardinality. Each subspace is then explored with a branch-and-bound algorithm.

Our work provides different relevant contributions. From the theoretical point of view, the importance of core items and of variable reduced costs is strongly emphasized. We believe that the use of such concepts deserves to be further explored in the context of more general integer programming problems. From the computational point of view, CORAL is shown to be, on average, a very efficient method and more stable with respect to the branch-and-bound algorithm proposed by Vimont et al. (2008) and with respect to CPLEX 10. In particular, on instances with few constraints, and independently of the number of items, CORAL is definitely the best method proposed in the literature. On instances with a larger number of constraints, CPLEX was more efficient provided that the number of items was small (no more than 100 items). Finally, CORAL was able to improve six best-known values for the class of instances with 250 items and 30 constraints in the OR-LIBRARY set (Chu and Beasley 1998).

The rest of this paper is organized as follows. In §2, we present the general scheme of the exact approach used to solve the MKP and analyze its main features. Section 3 is devoted to the restricted core problem and to the algorithm used to solve it. Finally, in §4, extensive computational results on known benchmark problems are presented and discussed. Conclusions are drawn in §5.

2. The Exact Algorithm

The algorithm attempts to rapidly find high-quality feasible solutions. This is achieved by solving subproblems limited to subsets of variables. Each subproblem is solved by a two-step procedure. The first step is a recursive procedure aimed at fixing to the optimal value some of the problem variables or at improving the lower bound on the optimum of the problem. When the number of variables not yet fixed to their optimal value decreases below a given threshold, the subproblem limited to the remaining set of variables is solved with an ad hoc algorithm that splits the solution space into several subspaces, each containing solutions of a given cardinality. Each subspace is optimally explored with a branch-and-bound algorithm.

During the algorithm, the set of items for which the associated variable is not fixed to a value constitutes the set of core items. The set of *core* items is the subset of items for which we find it difficult to decide whether the items will belong to the optimal solution. To identify the core set, an efficiency value is associated to each item. Efficiency is a measure of the likelihood that the variable associated to the item can be set to its optimal value. The *restricted set of core items* is a set that has a size lower than a given threshold. The MKP limited to the restricted set of core items is called the *restricted core* problem. The optimal solution of the restricted core problem is one of the most complex parts of the algorithm. Because the idea of a set of core items is key to the overall approach, we call it CORAL for CORe ALgorithm.

The idea of solving a *core* problem is not new in the family of knapsack problems. The knapsack core concept was initially proposed by Balas and Zemel (1980) for the 0-1 knapsack problem and then analyzed by Pisinger (1999), and it was more recently generalized to 0-1 integer programming by Huston et al. (2008). Puchinger et al. (2010) extended such a concept to the multidimensional case and evaluated the effectiveness of different efficiency measures. We use a similar idea of a core set but introduce several differences with respect to the previous approach—first, in the way we measure the item efficiency. We define item efficiency as the absolute value of the reduced cost of the associated variable in the continuous relaxation problem.

In this section we present the general structure and main features of CORAL. Unless explicitly mentioned, we will denote by LB and UB the best current lower bound and upper bound on the optimal solution value, respectively. Moreover, we indicate as MKP(N) the problem formulated on the original set N of items.

2.1. Two Known Results

CORAL is centered on two main known results. The first result is on problem reduction. Assume some feasible solution $\bar{x}$ is known with an objective function value $\bar{z}$. Let z_k be the optimal solution value of the MKP, where the variable x_k has been set to a value a. Then,

PROPOSITION 1. *If $z_k \le \bar{z}$, then either $\bar{x}$ is an optimal solution of the MKP or $x_k = 1 - a$ in every optimal solution.*

The second result is a logic cut introduced for the MKP by Oliva et al. (2001). Given a feasible solution for the MKP with a value LB and the optimal solution value UB of its continuous relaxation, the following relation holds:

$$\mathrm{UB} + \sum_{j\in N^-} b_j x_j - \sum_{j\in N^+} b_j(1-x_j) + \sum_{i\in M} u_i s_i \ge \mathrm{LB}, \qquad (4)$$

where s_i are the slack variables, and b_j and u_i are the reduced costs associated to nonbasic variables. Set N^- (set N^+) is the set of items associated to nonbasic variables with a value of 0 (a value of 1). Thus, b_j, $j \in N^+$, are positive reduced costs, whereas b_j, $j \in N^-$, are negative ones. The efficiency value associated to each item j, $j \in N$, is $|b_j|$.

Since u_i are the reduced costs of the slack variables, we know that $u_i \le 0$ and $s_i \ge 0$. Thus, the quantity $\sum_{i\in M} u_i s_i$ can be removed, giving rise to the final form of the logic cut:

$$\sum_{j\in N^+} b_j(1-x_j) + \sum_{j\in N^-} |b_j| x_j \le \mathrm{UB} - \mathrm{LB}. \qquad (5)$$

We identify such an inequality as the *reduced costs inequality*. Because inequality (5) must be satisfied by any optimal solution, we will use it as follows. A nonbasic variable can change its value with respect to the optimal value it has in the continuous relaxation only if its absolute reduced cost value is not greater than UB − LB.

2.2. Structure of CORAL

We describe here the structure of CORAL. The procedure starts with an *initialization phase* where a set of nonbasic variables is fixed to the optimal value of the continuous relaxation, and the MKP limited to the other variables is optimally solved (subproblem MKP($N, \bar{N}_0, \bar{N}_1$), where $\bar{N}_0$ and $\bar{N}_1$ are the sets of variables with values fixed to 0 and to 1, respectively). If we optimally solve a subproblem MKP($N, \bar{N}_0, \bar{N}_1$) limited to a subset of variables and x_s is the variable (not included in the subset) that has a minimum absolute value of the reduced cost, then $|b_s| > (\mathrm{UB} - \mathrm{LB})$ is an optimality condition. This is the *optimality condition* CORAL uses.

Then, the procedure continues with an *iterative phase*, where subproblems are optimally solved to improve the current lower bound and make the optimality condition stronger. Each subproblem is limited to a subset of variables (core items), and the value of the other variables is fixed. At each iteration, a new item is added to the set of core items. Thus, the number of variables of the subproblems monotonically increases.

CORAL
INPUT: Problem MKP(N).
OUTPUT: Optimal solution x^* and its value z^*.

Initialization Phase

1. Compute an initial lower bound LB through a *Heuristic*.
2. Compute an initial upper bound UB through the optimal solution of the continuous relaxation of problem MKP(N).
3. Sort the items in nonincreasing order of the absolute value of their reduced cost.
4. Construct the initial set of core items N_{core} by selecting the last C items with $C \ge m$; let $\bar{N} = N \backslash N_{\text{core}}$ be the remaining set of items.

5. Set each variable $x_s, s \in \bar{N}$, to the optimal value of the continuous relaxation; let $\bar{N}_0$ and $\bar{N}_1$ be the set of variables set to 0 and to 1, respectively, with $\bar{N} = \bar{N}_0 \cup \bar{N}_1$.

6. Solve to optimality the subproblem MKP$(N, \bar{N}_0, \bar{N}_1)$. Let $\bar{z}$ be the optimal value. If $\bar{z} + \sum_{j\in\bar{N}_1} p_j >$ LB, then update the lower bound LB.

Iterative Phase

1. Let $s := \arg\min_{j\in\bar{N}} |b_j|$. If $|b_s| \geq (\mathrm{UB} - \mathrm{LB})$, then the algorithm terminates, providing the optimal solution (x^*, z^*).

2. If $s \in \bar{N}_0$, then $\bar{N}_0 = \bar{N}_0 \backslash \{s\}$ and $\bar{N}_1 = \bar{N}_1 \cup \{s\}$; otherwise, $\bar{N}_1 = \bar{N}_1 \backslash \{s\}$ and $\bar{N}_0 = \bar{N}_0 \cup \{s\}$.

3. Solve to optimality the subproblem MKP$(N, \bar{N}_0, \bar{N}_1)$; let $\bar{z}$ be its optimal value.

4. If $\bar{z} + \sum_{j\in\bar{N}_1} p_j >$ LB, then update the lower bound LB and set $z^* =$ LB, with x^* as the corresponding integer solution.

5. $N_{\mathrm{core}} = N_{\mathrm{core}} \cup \{s\}$; $\bar{N} = \bar{N} \backslash \{s\}$. Go to step 1 of this phase.

In the initialization phase, after computing a lower and an upper bound LB and UB for the original problem MKP(N), the algorithm sorts the items for efficiency value (step 3). According to the absolute value of its reduced cost, the algorithm decides whether a variable has to be assigned to the initial core set. Different rules can be worked out to identify the initial set N_{core} in step 4. We simply select the last C items in the sorting of step 3, where C is a given parameter. The rationale is that we expect that the larger the absolute value of a reduced cost, the higher the likelihood that the corresponding variable will be set to 1 or to 0 in the optimal solution if its reduced cost is positive or negative, respectively. Moreover, because in an MKP with m constraints at most m variables are fractional in the optimal solution of the continuous relaxation, we set $C \geq m$. Once decided which items will make part of the initial core set and the value to which all the other variables will be set (step 5), the subproblem MKP$(N, \bar{N}_0, \bar{N}_1)$ is optimally solved. If its optimal value plus the sum of the profits of variables already set to 1 ($\bar{N}_1$) is greater than LB, the latter value is updated (step 6).

In the iterative phase, CORAL can be seen as a continuous *lower bound improvement* procedure that works through the optimal solution of subproblems MKP$(N, \bar{N}_0, \bar{N}_1)$, where each variable of the set $\bar{N}_0$ is fixed to 0 and each variable of the set $\bar{N}_1$ is fixed to 1, until the optimality condition is reached. At each iteration, the item outside the core set with minimum efficiency value is selected to enter the core set (step 1). If the optimality condition is satisfied, the algorithm terminates. If this is not the case, variable x_s is temporarily set to 1 if s belongs to $\bar{N}_0$ and set to 0 otherwise. In this way, the size of the core set remains unchanged, although subproblem MKP$(N, \bar{N}_0, \bar{N}_1)$ changes. The optimal solution of subproblem *MKP*$(N, \bar{N}_0, \bar{N}_1)$ is then used to possibly update the current lower bound LB (step 4). Finally, item s is added to the core set (step 5), and the phase is repeated. At each iteration, the size of the subproblem increases by 1. This means that the algorithm may terminate considering all problem variables. However, the solution of larger subproblems is made easier by the lower bound improvement.

Each subproblem MKP$(N, \bar{N}_0, \bar{N}_1)$ is optimally solved through the following two-step procedure:

- the reduction of the set of items through *variable fixing*, and then
- the optimal solution of the restricted core problem.

The first step aims at either fixing to their optimal value subsets of variables or at finding new best solutions and thus at improving the lower bound. In turn, the improvement of the lower bound allows the fixing of more variables to their optimal value. The second step starts when the number of variables in the set of core items decreases below a given threshold. Then, the restricted core problem is optimally solved with an ad hoc algorithm. These two steps will be analyzed in the next two sections.

We have also implemented a variant of CORAL algorithm (named CORAL with Cardinality) that computes an estimate on the minimum $k_{\min}$ and the maximum $k_{\max}$ number of items making part of the optimal solution of problem MKP(N) and generates $k_{\max} - k_{\min} + 1$ different subproblems. Specifically, subproblem k, $k_{\min} \leq k \leq k_{\max}$, is obtained from MKP$(N)$ by adding the cardinality constraint $\sum_{j=1}^{n} x_j = k$. Each subproblem is optimally solved by using CORAL. As we will see in §4, there are situations where this variant may be more efficient.

2.3. Variable Fixing

The first step of the procedure used to optimally solve any subproblem MKP$(N, \bar{N}_0, \bar{N}_1)$ aims at fixing variables to their provably optimal value and stops when the size of the set of the core items reaches a value lower than or equal to a parameter $n_{\max}$. We call this first step procedure VariableFixing( ). Figure 1 shows its pseudocode.

The procedure receives as input the subproblem MKP$(N, \bar{N}_0, \bar{N}_1)$ and a lower bound $\overline{\mathrm{LB}}$, computed as $\mathrm{LB} - \sum_{j\in\bar{N}_1} p_j$, and provides as output a restricted core problem with a number of variables lower than or equal to $n_{\max}$ as well as a possibly improved lower bound value LB′. Notice that if the optimal solution value of the subproblem is not greater than $\overline{\mathrm{LB}}$, then it will not improve the lower bound LB on the problem MKP(N).

INPUT:
lower bound $\overline{\text{LB}}$; problem MKP$(N, \bar{N}_0, \bar{N}_1)$;
OUTPUT:
lower bound LB$'$; *restricted core* problem of MKP$(N, \bar{N}_0, \bar{N}_1)$;
VARIABLEFIXING( ):
Set LB$' := \overline{\text{LB}}$;
Compute UB$'$ as the continuous relaxation value of
  MKP$(N, \bar{N}_0, \bar{N}_1)$;
Apply *direct fixing*: compute set D;
while $(|N_{\text{core}}| - |D| > n_{\max})$ **do**
  Let s be the item such that $|b_s| = \max_{j\in N_{\text{core}}\setminus D}\{|b_j|\}$;
  if $b_s > 0$, then $x_s := 0$, **otherwise** $x_s := 1$;
  Apply *temporary fixing*: compute set I;
  Set $F_1 := \{j \in D\cup I\cup\{s\}: x_j = 1\}$ and $F_0 := \{j \in D\cup I\cup\{s\}: x_j = 0\}$;
  if size of MKP$(N, \bar{N}_0\cup F_0, \bar{N}_1\cup F_1)$ is greater than $n_{\max}$ **then**
    Apply VARIABLEFIXING( );
    Let z' be its optimal solution value; set $\bar{z} := z'$;
  else
    Solve the *restricted core* problem;
    Let z'' be its optimal solution value; set $\bar{z} := z''$;
  end if
  if $\bar{z} + \sum_{j\in F_1} p_j > \text{LB}'$ **then**
    set LB$' = \bar{z} + \sum_{j\in F_1} p_j$;
  else
    set $x_s := 1 - x_s$ and $D := D\cup\{s\}$;
  end if
end while
Let $D_1 := \{j \in D: x_j = 1\}$ and $D_0 := \{j \in D: x_j = 0\}$;
Set $\bar{N}_0 := \bar{N}_0 \cup D_0$ and $\bar{N}_1 := \bar{N}_1 \cup D_1$.

Figure 1 **Pseudocode of VARIABLEFIXING( ) Procedure**

The procedure starts by initializing LB$'$ with $\overline{\text{LB}}$ and computing an upper bound UB$'$ corresponding to the optimal solution value of the continuous relaxation of the subproblem MKP$(N, \bar{N}_0, \bar{N}_1)$. The procedure called *direct fixing* uses the value of the optimal solution of the current problem continuous relaxation to possibly fix some problem variables by means of reduced costs inequality (5). If the absolute value of the reduced cost of a nonbasic variable exceeds UB$'$ − LB$'$, then the variable is optimally set to the value taken in the continuous relaxation solution. We define D as the set of variables with fixed value. After the direct fixing, if the number of remaining variables is greater than $n_{\max}$, a main routine is run to fix the values of the other $|N_{\text{core}}| - |D| - n_{\max}$ variables. Among the variables whose absolute reduced costs coefficient is lower than UB$'$ − LB$'$, the one with largest absolute reduced cost is considered, and its value is temporarily fixed to 1 if it was set to 0 by the continuous relaxation and is set to 0 otherwise. Let this variable be x_s. The temporary value assigned to this variable modifies inequality (5), and this may imply, through the direct fixing procedure described above, the fixing of the value of some additional nonbasic variables (we call this process *temporary fixing*). We define I as the set of variables with value fixed by the temporary fixing.

We indicate by F_1 and F_0 the sets of variables fixed to 1 and 0, respectively, by the direct and temporary fixing. Then, a new subproblem, MKP$(N, \bar{N}_0\cup F_0, \bar{N}_1\cup F_1)$, is constructed. If such subproblem has a size greater than $n_{\max}$, the procedure VARIABLEFIXING( ) is recursively applied; otherwise, the restricted core problem is solved.

The optimal solution of the subproblem can be of two basic different types: the value of the optimal solution $\bar{z}$ does not allow us to improve the best lower bound available LB$'$ or the value of the lower bound is improved. In the former case, the value temporarily assigned to variable x_s cannot produce any solution better than LB$'$ and, in turn, better than LB. Thus, x_s is set to the complement to 1 of its current value, the set D is updated with item s, and the successive nonbasic variable with largest absolute reduced cost value is considered; the procedure described above is repeated. In the latter case, the value of x_s cannot be set, but the lower bound LB$'$ can be improved. This implies that the reduced costs inequality (5) becomes tighter, and this will help the subsequent part of the solution of the subproblem. We hope that this will allow us to fix to the optimal value more nonbasic variables of the subproblem. The size of the set of core items is not reduced, the successive nonbasic variable is considered, and the procedure described above is repeated.

Finally, when the main routine is terminated, the sets $\bar{N}_0$ and $\bar{N}_1$ are updated with the items in set D. A possible better lower bound LB$'$ and the resulting restricted core problem containing a number of items less than or equal to $n_{\max}$ are returned.

The innovative aspect of this procedure is its recursive nature. Notice that the termination of the recursion is guaranteed by the fact that each problem in the recursion tree has a number of variables lower than that of the father problem by at least one unit (the variable temporarily fixed). In the following, we describe an example, illustrated through Figures 2(a)–(e), to clarify how the procedure VARIABLEFIXING( ) works.

Problem P is the initial subproblem to be solved. After the direct fixing, the size of the problem is still greater than $n_{\max}$. Thus, given the optimal solution of the continuous relaxation, the variable with the largest absolute reduced cost that does not violate constraint (5) (say, x_{j_1}) is selected, and its value is set to r_{j_1} (corresponding to the complement to 1 of the value the variable has taken in the optimal solution of the continuous relaxation). The resulting problem, P_1, obtained after temporary fixing, turns out to have a size lower than $n_{\max}$ and can be directly solved (Figure 2(a)). The resulting solution value is not better than the current lower bound, and therefore we can fix $x_{j_1} = 1 - r_{j_1}$. The original problem P has now one more variable with a fixed value. Nevertheless, it still has a size greater than $n_{\max}$, and thus the subsequent variable with largest absolute reduced cost is considered (say, x_{j_2}), and its value is temporarily set to r_{j_2}, giving rise to problem P_2 (Figure 2(b)).

Problem P_2 needs to be further processed through VariableFixing( ) (the first level of recursion) because its size is greater than $n_{\max}$. We recompute the constraint (5) using the reduced costs and the upper bound provided by the optimal solution of the continuous relaxation of problem P_2, and we fix its variables through direct and temporary fixing. Again, some variables of P_2 are set by direct fixing. Because the resulting problem size is still greater than $n_{\max}$, then the variable with largest reduced cost is selected (x_{h_1}) and temporarily set to r_{h_1}. The resulting problem, P_3, still has a number of variables greater than $n_{\max}$, thus requiring a new call to the VariableFixing( ) procedure (the second level of recursion). This gives rise to three subproblems P_4, P_5, and P_6, all with sizes lower than $n_{\max}$ and all solved to optimality. However, whereas the optimal solutions of P_4 and P_6 do not provide any lower bound improvement, the problem P_5 solution improves the lower bound value (Figure 2(c)).

The solution of P_4, P_5, and P_6 allows us to reduce the size of problem P_3, which can now be solved to optimality. Unfortunately, the resulting solution does not provide a lower bound improvement. This implies that, at the higher level of the recursion, $x_{h_1} = 1 - r_{h_1}$ (Figure 2(d)). At this point, problem P_2 has one more variable fixed but still has a size greater than $n_{\max}$. We then set $x_{h_2} = r_{h_2}$ and obtain problem P_7, which, thanks to the temporary fixing, has less than $n_{\max}$ variables and is optimally solved, allowing us to set $x_{h_2} = 1 - r_{h_2}$. Now both x_{h_1} and x_{h_2} are fixed, and at the higher level of the recursion, this implies that problem P_2 can now be solved, finally setting $x_{j_2} = 1 - r_{j_2}$. The initial subproblem now has the required size, and the algorithm ends (Figure 2(e)).

The feasible solution improving the lower bound, if any is found, will be saved only when the restricted core problem of the root node will be solved.

3. The Restricted Core Problem

The *restricted core* problem is a subproblem MKP$(N, \bar{N}_0, \bar{N}_1)$, where the size of the core items set N_{core} is lower than or equal to $n_{\max}$. The problem is formulated on the items in N_{core} by reducing the right-hand side of each constraint of the sum of the weights of variables in $\bar{N}_1$:

$$\max \sum_{j \in N_{\text{core}}}^{n} p_j x_j \tag{6}$$

$$\sum_{j \in N_{\text{core}}} w_{ij} x_j \le c_i - \sum_{j \in \bar{N}_1} w_{ij} \quad i \in M, \tag{7}$$

$$x_j \in \{0, 1\} \quad j \in N_{\text{core}}. \tag{8}$$

By combining the optimal solution of the restricted core problem with the partial solution defined by $\bar{N}_0$ and $\bar{N}_1$, we obtain a feasible solution for the original problem.

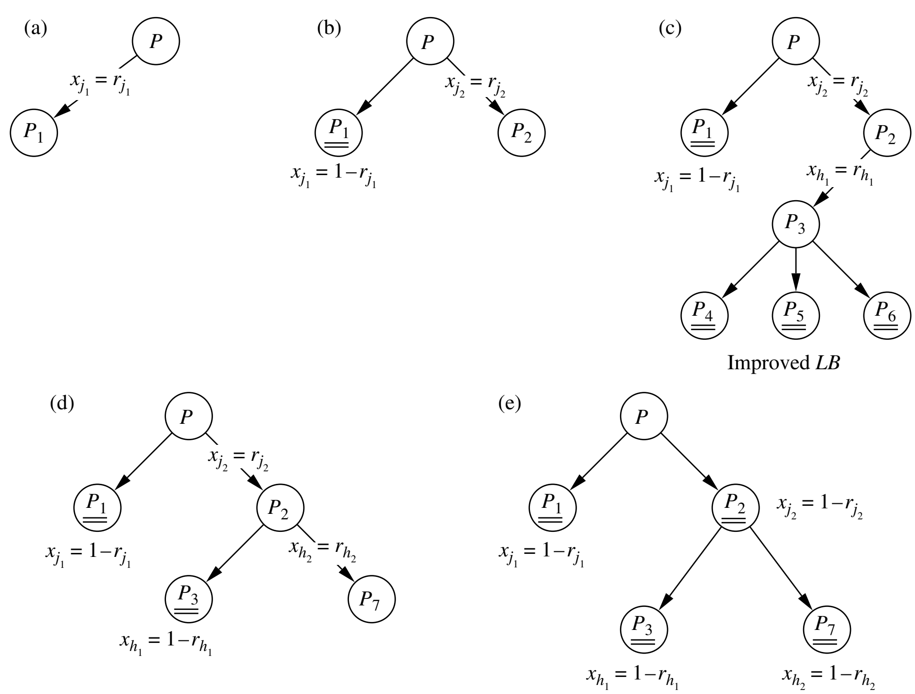

Figure 2 VariableFixing( ): An Example

3.1. Solution of the Restricted Core Problem

The exact algorithm for the solution of the restricted core problem partitions the solution space into subspaces, each one containing solutions with the same cardinality. Each subspace is explored with a branch-and-bound algorithm. The binary structure of each branch-and-bound tree is determined by the inclusion or exclusion of an item. Thus, each tree level is associated with an item, for a maximum number of levels equal to $|N_{\text{core}}|$ in each tree. At each node, the branching scheme selects the not-yet-fixed item j having the maximum profit and generates two descendant nodes by fixing x_j to 1 and 0. In a tree searched for cardinality k, we define *complete* as a feasible solution containing k items and *partial* as a feasible solution containing a number of items lower than k. The pseudocode description of the algorithm is provided in Figure 3. The procedure receives as input the restricted core set N_{core} and a lower bound value LB_{core} for the restricted core problem and provides as output the optimal value z^*_{core} if a solution x^*_{core} (if any) with value better than LB_{core} has been found. LB_{core} is obtained by reducing the best incumbent feasible solution value LB for the original problem by the sum of values of the items in $\bar{N}_1$; i.e., $\text{LB}_{\text{core}} := \text{LB} - \sum_{j\in\bar{N}_1} p_j$. This means that LB_{core} may not correspond to the value of a feasible solution for the restricted core problem. Indeed, LB_{core} is used as a cut-off value representing a threshold under which the optimal solution value of the restricted core problem cannot improve the lower bound LB.

To limit the number of binary trees to explore, we estimate the minimum and maximum number of items in the restricted core problem solution. Let us define such values as $\bar{k}_{\min}$ and $\bar{k}_{\max}$, respectively. Denoting by k^*_{core} the number of items in the optimal solution of the restricted core problem, we have that $\bar{k}_{\min} \le k^*_{\text{core}} \le \bar{k}_{\max}$. The value of $\bar{k}_{\min}$ is computed by rounding up the optimal solution value of the following linear programming (LP) problem:

$$\min \sum_{j\in N_{\text{core}}} x_j \tag{9}$$

$$\sum_{j\in N_{\text{core}}} p_j x_j \ge \text{LB}_{\text{core}} + 1, \tag{10}$$

$$\sum_{j\in N_{\text{core}}} w_{ij} x_j \le c_i - \sum_{j\in N_1} w_{ij} \quad i \in M, \tag{11}$$

$$0 \le x_j \le 1 \quad j \in N_{\text{core}}. \tag{12}$$

The value $\bar{k}_{\max}$ is obtained by solving a maximization instead of a minimization problem and rounding down the optimal solution value.

The procedure *SearchTree*(k) searches the space of solutions containing exactly k items. To speed up the search, the procedure makes use of conditions which prune parts of the search tree. We have implemented four different types of conditions to be applied in any node of the binary tree. If a pruning condition is satisfied, the subtree stemming from the current node is fathomed. Conditions are applied in sequence to each node.

```
INPUT: LB_core, N_core.
OUTPUT: optimal solution x*_core (if any) with value z*_core.
  Sort the items in N_core in nonincreasing order of their values p_j;
  Compute k̄_min and k̄_max;
  Set k := k̄_min and z*_core := LB_core;
  while (k ≤ k̄_max) do
     Apply SearchTree(k) to the binary tree with cardinality k;
     Let x^(k) be the optimal solution and z^(k) its value;
     If z^(k) > z*_core then z*_core := z^(k) and x*_core := x^(k);
  end while
```

Figure 3 Pseudocode of the Exact Algorithm for the Restricted Core Problem

In the following, we assume to be in a node of the binary tree where the associated partial solution has value $\bar{z}$ and cardinality equal to $\bar{k}$. Let s be the item that has to be added next to the current partial solution. z^*_{core} provides the value of the best incumbent solution (best lower bound value). As specified in Figure 3, items are sorted in nonincreasing order of their value p_j.

Condition 1 (Value). Define as $P(s, q)$ the sum of the values of the q items following item s. The condition is satisfied when

$$\bar{z} + p_s + P(s, k - \bar{k} - 1) < z^*_{\text{core}}. \tag{13}$$

The first member of the inequality is equal to the maximum value we can get for a complete solution obtained from the current partial solution by adding item s. If such value is lower than z^*_{core}, no complete solution can be obtained from the current partial one with value greater than z^*_{core}.

Condition 2 (Total Weight). Let $W_j = \sum_{i=1}^m w_{ij}$ be the total weight associated with item j. Define $W(s, q)$ as the sum of the total weights of the q items with the smallest W_j values and such that $p_j \le p_s$. Let $\overline{W}$ be the sum of the total weights of items selected in the current partial solution, and $c = \sum_{i=1}^m c_i$. The condition is effective if

$$\overline{W} + W_s + W(s, k - \bar{k} - 1) > c. \tag{14}$$

The first member of the inequality is equal to the minimum weight we can get for a complete solution obtained from the current partial solution by adding item s. If the resulting weight is greater than c, no feasible solution can be created by adding item s to the current partial solution.

CONDITION 3 (REDUCED COSTS). Define H^- and H^+ as the set of items belonging to the current partial solution and whose reduced costs in the optimal solution of the continuous relaxation are negative and positive, respectively. The condition is valid if

$$|b_s| > (\mathrm{UB}_{\mathrm{core}} - z^*_{\mathrm{core}}) - \sum_{j=1}^{H^-} |b_j| - \sum_{j=1}^{H^+} b_j, \tag{15}$$

where $\mathrm{UB}_{\mathrm{core}}$ is the optimal solution value of the continuous relaxation of the restricted core problem.

CONDITION 4 (WEIGHT). Let H be the set of the items selected in the current partial solution including $\bar{N}_1$. We indicate as $V_i(s, q)$ the sum of the q smallest w_{ij} values in constraint i such that $p_j \le p_s$. The condition on constraint i is effective if the following inequality is satisfied:

$$\sum_{j \in H} w_{ij} + w_{is} + V_i(s, k - \bar{k} - 1) > c_i. \tag{16}$$

It establishes that a partial solution has to be cut off if it will never give rise to a feasible complete solution.

The four conditions have different effects in terms of the size of the pruned subtree. Because items are sorted by value, the pruning condition on value (Condition 1), when effective, is usually quite deep. For instance, let $n = 14$ and $k = 7$, and let us assume that the current partial solution has selected the items $1, 2, 4, 8$, and 10, whereas 11 is the next item to be inserted. If Condition 1 is effective, the next partial solution to start with is $1, 2$, with 5 the item to be inserted next. If Condition 2 is effective, the search is restarted from the current partial solution but considering item $s + l$ as next candidate to be added, where $l = \min\{h \mid W_{s+h} < W_s\}$. Condition 3 makes use of the reduced costs constraint (5) to eliminate partial solutions. If this condition is effective, we evaluate item $s + 1$ as the next item to be added to the current partial solution. Finally, Condition 4 is effective if constraint (16) is satisfied for at least one constraint, i, $i = 1, \ldots, m$. In such cases, we consider $s + 1$ as the next item to enter the current partial solution.

The procedure *SearchTree*(k) applies the pruning conditions in the order given before. The first condition that results to be effective defines the partial solution to be analyzed next. Notice that Condition 2 is the relaxed form of Condition 4 and thus requires less computational time to be checked with respect to the latter one. For this reason, pruning condition on total weight (Condition 2) is applied first: this might avoid the application of the other and more cumbersome condition. If no pruning condition is satisfied, the current item s is added to the partial solution, and the search is restarted with the evaluation of the insertion of the item $s + 1$. As soon as the whole tree has been visited, the best feasible complete solution $x^{(k)}$ is provided as output, and its value $z^{(k)}$ is used to possibly update the best value z^*_{core}. Then if $k + 1 \le \bar{k}_{\max}$, a new binary tree is taken into account. When the whole sequence of binary trees has been analyzed, the best feasible solution found (if any) is provided as the optimal solution for the restricted core problem.

Usually, $\bar{k}_{\min}$ and $\bar{k}_{\max}$ have values quite far from the optimal cardinality. After some preliminary results, we have decided to set the initial cardinality equal to $k_{av} + 1$, where $k_{av} = \lceil (\bar{k}_{\min} + \bar{k}_{\max})/2 \rceil$. Then the trees with cardinality greater than or equal to $k_{av} + 1$ are analyzed first, with items sorted according to the weights. Then, trees with cardinality lower than k_{av} are analyzed starting with $k = k_{av}$ and items sorted by values.

4. Experimental Analysis

CORAL was tested on two different classes of benchmark instances. The first class consists of 270 large MKP instances used by Chu and Beasley (1998) and made publicly available in the OR-LIBRARY at http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapinfo.html. This data set was constructed by the authors using the procedure suggested by Fréville and Plateau (1994) to generate correlated instances. The number of constraints m was set to 5, 10, and 30, and the number of variables n was set to 100, 250, and 500. There are 30 instances for each combination of m and n, giving a total of 270 instances. The w_{ij} are integer numbers uniformly drawn in $[0, 1{,}000]$. For each combination (n, m), the constraint capacities are computed as $c_i = \alpha \sum_{j=1}^{n} w_{ij}$, where α is a tightness ratio. For the first 10 instances, $\alpha = 0.25$; for the next 10 instances, $\alpha = 0.5$; and for the remaining ones, $\alpha = 0.75$. The objective function coefficients (p_js) were correlated to w_{ij} as follows: $p_j = \sum_{i \in M} w_{ij}/m + 500 d_j$, $j \in N$, where d_j is uniformly random in U(0, 1). In general, correlated instances are more difficult to solve than uncorrelated ones. The second class considers the first 7 instances out of the 11 huge instances proposed by Glover and Kochenberger (1996). This set consists of correlated and uniform randomly generated instances involving up to 2,500 items and 100 constraints. Data set can be downloaded at http://hces.bus.olemiss.edu/tools.html.

Algorithm CORAL was written in Java 5.0. All computational tests were conducted on a Pentium IV with 3 GHz and 2 GB of RAM running Windows XP Professional Service Pack 2. For the smallest benchmark instances, the initial lower bound (see step 1 of the initialization phase) was computed using a greedy algorithm (items sorted in nonincreasing order of the ratio profit/(sum of weights)). For more complex instances, the initial solution value was provided by the heuristic framework *kernel search* (KS) (Angelelli et al. 2010).

According to the value assigned to some parameters, the KS gives rise to different heuristic implementations. Specifically, we used the heuristic defined as *Fixed-Bucket*-I-(1). Details on this procedure and other implementations of the KS when applied to the MKP can be found in Angelelli et al. (2010). Solution values found by KS on the Chu and Beasley (1998) instances and by CORAL on both classes of instances can be found at http://www.unibs.it/on-line/dmq/Home/Personale/articolo2398.html.

4.1. Computational Results: Chu and Beasley's (1998) Instances

After some preliminary testing, we decided to set the initial size C of the core set and the maximum size $n_{\max}$ for the restricted core problems both equal to 30 for all the tested instances of this data set.

At the time of these experiments, classes of instances 5.100 (i.e., with $m=5$ and $n=100$) and 10.100 had already been solved to optimality (see Chu and Beasley 1998). Moreover, Vimont et al. (2008) found the optimal solutions for the sets of instances 5.250, 5.500, and 10.250 using a Pentium 4 with 3.2 GHz and 1 GB of RAM, which is comparable to the computer we used. Afterwards, in a more recent paper, Boussier et al. (2008) provided the optimum for the class of instances 10.500. We identify these exact methods as VBV and BVVHM from the names of their authors, respectively. In the following, we show the results obtained by CORAL as well as those by CPLEX 10.0 that are used as a unique benchmark whenever no solution values from other methods are available.

We applied the CORAL with Cardinality constraint only to instances where $m=5$ (with the exclusion of instances 5.100) and to instances 10.100. On average, only in instances with very few constraints did the introduction of the cardinality constraint seem to help in reducing the continuous relaxation upper bound value and in generating larger (in absolute value) reduced costs to make constraint (5) more effective. Table 1 provides the percentage deviation between the average computational time (out of 10 instances with the same tightness ratio) required by CORAL with respect to the average time required by its variant. A negative deviation means that CORAL has an average computing time lower with respect to its variant with cardinality. Notice that with the only exclusion of instances 5.100, CORAL with Cardinality is more efficient in these instances.

Table 1 CORAL vs. CORAL with Cardinality: Deviation of Average Computational Times (in Percent)

α	5.100	5.250	5.500	10.100
0.25	−31.33	23.04	25.38	17.75
0.50	−29.70	23.18	11.20	17.60
0.75	−6.52	12.61	15.64	3.87

Tables 2–6 provide the results for the first five classes of instances. In all such instances, the previously mentioned greedy heuristic was used to find an initial lower bound. All tables have the same structure. The first column provides the instance solved. The second and third columns show the value of the optimal solution (Opt) and the cardinality of the optimal solution (#) as determined by our algorithm. Then, the computational times required to find the optimal value by our algorithm (CORAL), by the Vimont et al. (2008) approach (VBV), and finally by CPLEX are shown. Times are expressed in seconds with two decimal digits except for VBV, because the times are provided as integers by the authors. Tables 2, 5, and 6 do not contain column VBV because no results are available for such an algorithm. It is worth noting that the set of instances 30.100 has never been solved to optimality before now.

Table 2 Chu and Beasley (1998) Benchmark Instances Where $m=5$ and $n=100$: Optimal Values

Problem	Opt	#	CORAL time (s)	CPLEX time (s)
5.100-00	24,381	29	2.81	6.14
5.100-01	24,274	29	2.20	2.66
5.100-02	23,551	29	3.48	4.95
5.100-03	23,534	28	4.59	19.81
5.100-04	23,991	30	1.95	6.23
5.100-05	24,613	30	1.62	2.50
5.100-06	25,591	31	0.78	0.64
5.100-07	23,410	28	2.56	2.53
5.100-08	24,216	28	1.22	4.09
5.100-09	24,411	28	1.67	4.28
Avg.			2.29	5.38
5.100-10	42,757	52	0.75	1.28
5.100-11	42,545	50	0.81	1.83
5.100-12	41,968	53	8.83	35.39
5.100-13	45,090	52	3.14	6.23
5.100-14	42,218	54	1.92	4.03
5.100-15	42,927	54	0.51	1.08
5.100-16	42,009	55	1.81	0.37
5.100-17	45,020	53	5.84	8.59
5.100-18	43,441	52	1.11	1.16
5.100-19	44,554	53	2.75	3.05
Avg.			2.75	6.30
5.100-20	59,822	78	0.75	0.59
5.100-21	62,081	77	0.42	0.67
5.100-22	59,802	77	0.78	2.06
5.100-23	60,479	76	1.33	3.44
5.100-24	61,091	78	0.98	1.42
5.100-25	58,959	76	2.06	2.58
5.100-26	61,538	76	0.59	1.20
5.100-27	61,520	76	1.58	1.02
5.100-28	59,453	76	0.55	0.59
5.100-29	59,965	76	5.64	4.09
Avg.			1.47	1.77
Total avg.			2.17	4.48

Table 3 Chu and Beasley (1998) Benchmark Instances Where $m = 5$ and $n = 250$: Optimal Values

Problem	Opt	#	CORAL time (s)	VBV time (s)	CPLEX time (s)
5.250-00	59,312	73	4.70	80	64.48
5.250-01	61,472	74	24.76	82	167.53
5.250-02	62,130	76	8.34	62	14.20
5.250-03	59,463	71	129.33	199	943.37
5.250-04	58,951	74	33.69	101	448.09
5.250-05	60,077	62	68.19	106	428.61
5.250-06	60,414	75	14.84	104	242.22
5.250-07	61,472	59	74.09	137	222.55
5.250-08	61,885	76	21.76	76	60.36
5.250-09	58,959	59	6.69	87	21.37
Avg.			38.64	103.40	261.28
5.250-10	109,109	133	38.84	111	209.44
5.250-11	109,841	116	13.27	99	26.44
5.250-12	108,508	116	27.95	75	140.66
5.250-13	109,383	133	59.55	110	219.70
5.250-14	110,720	130	108.12	116	833.22
5.250-15	110,256	115	37.92	96	207.67
5.250-16	109,040	121	19.61	82	147.72
5.250-17	109,042	122	61.52	109	462.87
5.250-18	109,971	131	79.77	105	187.11
5.250-19	107,058	115	49.61	96	253.52
Avg.			49.62	99.90	268.84
5.250-20	149,665	178	42.73	96	116.02
5.250-21	155,944	171	14.73	95	80.02
5.250-22	149,334	176	38.56	119	154.92
5.250-23	152,130	172	15.06	105	88.06
5.250-24	150,353	191	40.61	102	134.43
5.250-25	150,045	174	4.44	73	9.13
5.250-26	148,607	192	3.66	75	7.20
5.250-27	149,782	193	24.30	103	140.36
5.250-28	155,075	190	7.66	92	36.16
5.250-29	154,668	173	21.42	99	101.00
Avg.			21.32	95.90	86.73
Total avg.			36.53	99.73	205.61

Table 4 Chu and Beasley (1998) Benchmark Instances Where $m = 5$ and $n = 500$: Optimal Values

Problem	Opt	#	CORAL time (s)	VBV time (s)	CPLEX time (s)
5.500-00	120,148	147	838.953	1,331	9,495.08
5.500-01	117,879	148	189.94	1,034	2,112.22
5.500-02	121,131	144	353.91	1,112	5,188.23
5.500-03	120,804	149	293.98	630	6,390.36
5.500-04	122,319	147	345.80	558	2,860.30
5.500-05	122,024	153	614.06	932	3,657.51
5.500-06	119,127	145	850.91	851	8,498.31
5.500-07	120,568	150	274.66	804	3,631.08
5.500-08	121,586	149	765.78	1,172	10,424.12
5.500-09	120,717	150	414.42	2,783	6,641.16
Avg.			494.24	1,120.70	5,889.84
5.500-10	218,428	267	655.48	838	4,572.84
5.500-11	221,202	265	145.98	2,146	2,981.42
5.500-12	217,542	264	2,174.12	1,634	11,637.39
5.500-13	223,560	264	1,226.05	1,211	8,425.27
5.500-14	218,966	267	113.84	485	648.89
5.500-15	220,530	262	347.84	814	5,136.17
5.500-16	219,989	266	200.94	593	2,447.30
5.500-17	218,215	265	287.53	811	2,955.05
5.500-18	216,976	262	347.77	697	5,886.83
5.500-19	219,719	267	814.59	969	5,117.97
Avg.			631.42	1,019.80	4,980.91
5.500-20	295,828	383	70.52	767	218.31
5.500-21	308,086	384	392.58	674	3,744.97
5.500-22	299,796	385	141.09	783	746.42
5.500-23	306,480	384	220.70	620	5,467.58
5.500-24	300,342	385	326.59	610	2,554.25
5.500-25	302,571	385	134.53	634	1,218.58
5.500-26	301,339	385	72.45	466	837.47
5.500-27	306,454	383	62.00	558	804.24
5.500-28	302,828	384	232.78	724	2,336.92
5.500-29	299,910	378	863.89	560	8,301.98
Avg.			251.71	639.60	2,623.07
Total avg.			459.12	926.70	4,497.94

In all instances where $m = 5$ (Tables 2–4), CORAL turns out to be extremely efficient. For instances 5.100, it takes half the time required by CPLEX on the same computer, although the average computational time for both algorithms is negligible, being of only few seconds. In larger instances ($n \geq 250$), the gap in computational times between CORAL and CPLEX increases. For instances where $n = 250$, CORAL has an average time of 37 seconds and requires almost one-third of the time required, on average, by VBV and almost one-sixth of the time used by CPLEX. For the instances where $n = 500$, the time required by CPLEX (which is, on average, equal to 4,498 seconds) is an order of magnitude larger than that of CORAL, which requires 459 seconds. The latter is half of the average time required by VBV.

In Table 5 we provide the results for instances 10.100. Such instances are still quite easy for both CPLEX and CORAL. The two algorithms show an average computational time of the same order of magnitude, although CPLEX is slightly better. It is worth noting that for both algorithms, these instances require a computational time by an order of magnitude higher compared with that required by instances 5.100. When the number of constraints is further increased to 30, while n is kept to 100, instances become more correlated and thus more difficult to solve (see Table 6). In such instances CPLEX shows an average time of 774 seconds, whereas CORAL requires more than twice that time. Again, when the number of items is small, CPLEX is the best approach. Nevertheless, if the time in CPU seconds to find the optimal integer solution without proving optimality (time to best) is taken into account, the average performance of the two methods is very close.

In the most complex sets of instances (Tables 7–9), the solution value found by the kernel search heuristic was used as lower bound for both CPLEX and CORAL. In such cases, we do not provide a feasible solution as input to the exact routines but simply

Table 5 Chu and Beasley (1998) Benchmark Instances Where $m = 10$ and $n = 100$: Optimal Values

Problem	Opt	#	CORAL time (s)	CPLEX time (s)
10.100-00	23,064	15	140.33	145.16
10.100-01	22,801	14	125.20	98.20
10.100-02	22,131	26	34.59	28.48
10.100-03	22,772	27	237.50	174.72
10.100-04	22,751	14	26.39	10.81
10.100-05	22,777	15	124.72	160.95
10.100-06	21,875	15	42.98	26.98
10.100-07	22,635	27	11.23	13.77
10.100-08	22,511	27	18.39	24.34
10.100-09	22,702	13	85.14	30.48
Avg.			84.65	71.39
10.100-10	41,395	51	101.00	53.09
10.100-11	42,344	40	29.31	20.36
10.100-12	42,401	51	151.89	51.81
10.100-13	45,624	54	54.91	69.33
10.100-14	41,884	38	34.11	19.83
10.100-15	42,995	36	153.83	71.94
10.100-16	43,574	52	125.55	100.99
10.100-17	42,970	52	43.11	34.3
10.100-18	42,212	41	117.89	53.83
10.100-19	41,207	53	154.39	60.78
Avg.			96.60	53.63
10.100-20	57,375	77	4.69	3.06
10.100-21	58,978	75	47.99	26.16
10.100-22	58,391	56	40.69	39.53
10.100-23	61,966	75	7.64	4.22
10.100-24	60,803	74	7.22	6.92
10.100-25	61,437	76	28.56	15.44
10.100-26	56,377	76	59.56	41.86
10.100-27	59,391	75	7.84	3.81
10.100-28	60,205	76	17.72	9.06
10.100-29	60,633	61	10.34	9.61
Avg.			23.23	15.97
Total avg.			68.16	46.99

Table 6 Chu and Beasley (1998) Benchmark Instances Where $m = 30$ and $n = 100$: Optimal Values

Problem	Opt	#	CORAL time (s)	CPLEX time (s)
30.100-00	21,946	24	1,925.20	136.19
30.100-01	21,716	25	1,838.91	609.81
30.100-02	20,754	24	1,898.69	262.56
30.100-03	21,464	24	2,456.08	422.80
30.100-04	21,844	24	1,923.98	1,368.55
30.100-05	22,176	24	1,807.25	1,715.55
30.100-06	21,799	25	2,351.39	1,614.36
30.100-07	21,397	24	2,093.11	1,131.33
30.100-08	22,525	24	2,258.81	3,953.88
30.100-09	20,983	24	1,866.47	166.91
Avg.			2,041.99	1,138.19
30.100-10	40,767	49	1,991.53	1,422.05
30.100-11	41,308	49	2,803.87	2,226.39
30.100-12	41,630	50	2,536.12	2,852.22
30.100-13	41,041	50	1,984.77	259.63
30.100-14	40,889	49	1,951.98	976.84
30.100-15	41,058	48	2,672.33	678.31
30.100-16	41,062	49	1,924.12	954.69
30.100-17	42,719	49	2,015.41	530.92
30.100-18	42,230	49	956.03	30.77
30.100-19	41,700	49	1,937.25	331.36
Avg.			2,077.34	1,026.32
30.100-20	57,494	73	1,206.81	82.25
30.100-21	60,027	73	1,925.02	215.98
30.100-22	58,052	74	1,829.09	363.16
30.100-23	60,776	74	496.36	35.25
30.100-24	58,884	74	680.78	41.53
30.100-25	60,011	74	1,898.81	66.92
30.100-26	58,132	74	1,735.55	145.13
30.100-27	59,064	74	1,812.34	72.36
30.100-28	58,975	74	1,907.95	250.86
30.100-29	60,603	73	1,899.86	296.45
Avg.			1,539.26	156.99
Total avg.			1,886.20	773.83

use the heuristic solution value as a cutoff for the objective function value. Specifically, we set the cutoff value equal to the feasible solution value decreased by 1. In this way, we let the exact routine find the feasible solution associated with the lower bound. For CPLEX this corresponds to setting parameter *CutLo* to such a value. In these tables an entry equal to "infeas" means that, given the cutoff value, no feasible solution was found within the predefined time threshold.

For each instance in the class 10.250 (see Table 7), CORAL always found an optimal solution as well as VBV, whereas, in many instances, CPLEX was unable to find an optimal solution after 10 hours (36,000 seconds) of computational time. For many of these instances, the algorithm VBV was extremely efficient. Nevertheless, on average, its performance was worse than that of CORAL. Moreover, the latter provides a more stable behavior.

On the largest instances (i.e., classes 10.500 and 30.250), we compared CORAL and CPLEX performances within a predefined running time threshold of five hours (Tables 8 and 9). If one algorithm meets the best-known solution value, the corresponding figure is shown in bold, and an asterisk indicates whether such a value was improved. Again, the entry "infeas" means that no feasible solution was found after five hours with a value greater than or equal to the value provided as the lower bound.

For Table 8 best-known values correspond to optimal solutions found by Boussier et al. (2008) (algorithm BVVHM) and are reported in column "Opt." The third column of Table 8 provides the computational time (in hours) required by BVVHM to prove optimality. Times are huge, requiring, on average, several days. Within five hours CORAL was able to find the optimal solution value (without proving optimality) in 19 out of 30 instances, whereas in 20 instances, CPLEX terminated without getting a feasible solution. Class 30.250 has not been solved to optimality. In Table 9 the column "Best-known value" provides

Table 7 **Chu and Beasley (1998) Benchmark Instances Where $m = 10$ and $n = 250$: Optimal Values**

Problem	Opt	#	CORAL time (s)	VBV time (s)	CPLEX time (s)
10.250-00	59,187	68	3,798.97	4,921	infeas
10.250-01	58,781	69	3,870.27	43,618	8,662.64
10.250-02	58,097	69	3,885.63	1,335	13,683.70
10.250-03	61,000	70	4,363.74	8,874	36,000.00
10.250-04	58,092	67	4,587.94	14,487	36,000.00
10.250-05	58,824	68	3,718.63	964	36,000.00
10.250-06	58,704	67	3,615.36	898	36,000.00
10.250-07	58,936	69	5,393.69	87,129	36,000.00
10.250-08	59,387	68	3,612.19	4,240	36,000.00
10.250-09	59,208	69	4,233.19	6,729	36,000.00
Avg.			4,107.96	17,319.50	30,482.93
10.250-10	110,913	129	4,906.66	4,772	36,000.00
10.250-11	108,717	127	3,881.36	7,216	36,000.00
10.250-12	108,932	128	4,085.38	3,192	36,000.00
10.250-13	110,086	131	3,741.52	14,871	36,000.00
10.250-14	108,485	128	3,637.99	2,122	36,000.00
10.250-15	110,845	130	7,192.63	5,770	36,000.00
10.250-16	106,077	129	3,762.77	7,604	36,000.00
10.250-17	106,686	128	3,923.56	5,322	36,000.00
10.250-18	109,829	127	3,767.11	4,566	36,000.00
10.250-19	106,723	131	3,752.92	1,121	36,000.00
Avg.			4,265.19	5,655.60	36,000.00
10.250-20	151,809	187	3,841.03	770	10,071.25
10.250-21	148,772	188	3,867.08	2,138	36,000.00
10.250-22	151,909	189	3,639.11	653	7,869.70
10.250-23	151,324	189	3,865.20	5,316	6,772.08
10.250-24	151,966	191	3,777.06	753	6,990.25
10.250-25	152,109	189	3,999.89	639	6,332.47
10.250-26	153,131	189	889.06	85	659.88
10.250-27	153,578	187	4,230.50	8,259	36,000.00
10.250-28	149,160	187	3,775.99	974	5,176.84
10.250-29	149,704	190	3,649.14	596	6,625.17
Avg.			3,553.41	2,018.30	12,249.76
Total avg.			3,975.52	8,331.13	26,098.07

the best feasible solution value available in the literature (the corresponding reference is indicated in parentheses). It is worth noting that CORAL always found the best-known solution. Finally, we also tested the largest classes of instances, 30.500. Because in five hours' computation time both CPLEX and CORAL were not able to improve known results, we do not report the data.

CORAL is a complex method combining different features. To better state the contribution provided by its several components, we computed some simple statistics. Tables 10–13 provide such results for all classes of instances optimally solved by CORAL, taking into account only three instances for each class (the first instance for each tightness ratio).

Table 10 is divided into two parts. In the first part, identified as subproblems, we provide the total number of solved subproblems MKP($N, \bar{N}_0, \bar{N}_1$) ("No."), the number of times the solution of a subproblem terminates with lower bound improvements

Table 8 **Chu and Beasley (1998) Benchmark Instances Where $m = 10$ and $n = 500$: CORAL and CPLEX Best Results Within a Computational Time Threshold of Five Hours**

Problem	Opt	BVVHM time (h)	CORAL (5 h)	CPLEX (5 h)
10.500-00	117,821	567.2	117,811	infeas
10.500-01	119,249	272.9	119,232	infeas
10.500-02	119,215	768.3	**119,215**	infeas
10.500-03	118,829	89.6	118,825	118,813
10.500-04	116,530	2,530.3	116,509	116,509
10.500-05	119,504	188	**119,504**	infeas
10.500-06	119,827	128	**119,827**	infeas
10.500-07	118,344	179.6	118,329	infeas
10.500-08	117,815	219.9	**117,815**	infeas
10.500-09	119,251	354.9	119,231	infeas
10.500-10	217,377	515.8	**217,377**	**217,377**
10.500-11	219,077	437.6	**219,077**	infeas
10.500-12	217,847	5.5	**217,847**	infeas
10.500-13	216,868	104.4	**216,868**	**216,868**
10.500-14	213,873	1,382.1	213,859	213,843
10.500-15	215,086	43.9	**215,086**	**215,086**
10.500-16	217,940	36.1	**217,940**	infeas
10.500-17	219,990	348.8	219,984	219,984
10.500-18	214,382	57.8	214,375	infeas
10.500-19	220,899	21.3	**220,899**	infeas
10.500-20	304,387	8.2	**304,387**	infeas
10.500-21	302,379	8.4	**302,379**	infeas
10.500-22	302,417	105.5	302,416	infeas
10.500-23	300,784	3.8	300,757	infeas
10.500-24	304,374	16.8	**304,374**	infeas
10.500-25	301,836	30.9	**301,836**	**301,836**
10.500-26	304,952	18.5	**304,952**	**304,952**
10.500-27	296,478	9.3	**296,478**	infeas
10.500-28	301,359	39.1	**301,359**	infeas
10.500-29	307,089	4.4	**307,089**	**307,089**

("Imp."), and the maximum size of a solved subproblem expressed as a percentage of the instance size ("Max size"). The second part refers to restricted core problems and provides their total number ("No."), the number of restricted core problems that provide an improvement ("Imp."), and the percentage of restricted core problems not solved because the value of their continuous relaxation optimal solution was lower than the value of the best integer solution available (current lower bound value) ("Nonsolved"). To further reduce the percentage of solved restricted core problems, we made some attempts to compute a better upper bound value with respect to the continuous relaxation. In particular, we implemented the global lifted cover inequalities as proposed by Kaparis and Letchford (2008). For small instances where the introduction of cover inequalities seems to be effective, however, the results are not very encouraging. In larger instances the cover inequalities increase computational times without improving the bounds. We think that this direction of research deserves to be better delved into.

Table 10 shows how the number of subproblems solved (number of iterations of the iterative phase)

Table 9 **Chu and Beasley Benchmark (1998) Instances Where $m = 30$ and $n = 250$: CORAL and CPLEX Best Results Within a Computational Time Threshold of Five Hours**

Problem	#	Best-known value	CORAL	CPLEX
30.250-00	63	56,824 (Angelelli et al. 2010, Vimont et al. 2008)	**56,842***	56,824
30.250-01	63	58,520 (Angelelli et al. 2010, Vimont et al. 2008)	**58,520**	infeas
30.250-02	63	56,553 (Angelelli et al. 2010, Vimont et al. 2008)	**56,614***	56,553
30.250-03	63	56,930 (Angelelli et al. 2010, Vimont et al. 2008)	**56,930**	**56,930**
30.250-04	63	56,629 (Angelelli et al. 2010, Vimont et al. 2008)	**56,629**	**56,629**
30.250-05	63	57,205 (Angelelli et al. 2010)	**57,205**	**57,205**
30.250-06	63	56,348 (Angelelli et al. 2010)	**56,357***	**56,357***
30.250-07	63	56,457 (Angelelli et al. 2010, Vimont et al. 2008)	**56,457**	**56,457**
30.250-08	63	57,442 (Angelelli et al. 2010, Vimont et al. 2008)	**57,442**	**57,442**
30.250-09	62	56,447 (Angelelli et al. 2010, Vimont et al. 2008)	**56,447**	**56,447**
30.250-10	125	107,770 (Vimont et al. 2008)	**107,770**	107,755
30.250-11	124	108,392 (Vimont et al. 2008)	**108,392**	108,379
30.250-12	124	106,440 (Angelelli et al. 2010)	**106,442***	**106,442***
30.250-13	124	106,876 (Vimont et al. 2008)	**106,876**	106,865
30.250-14	125	107,414 (Angelelli et al. 2010, Vimont et al. 2008)	**107,414**	**107,414**
30.250-15	126	107,271 (Vimont et al. 2008)	**107,271**	107,246
30.250-16	126	106,365 (Vimont et al. 2008)	**106,372***	106,365
30.250-17	124	104,014 (Angelelli et al. 2010)	**104,032***	104,024
30.250-18	123	106,835 (Vimont et al. 2008)	**106,835**	**106,835**
30.250-19	125	105,780 (Angelelli et al. 2010, Vimont et al. 2008)	**105,780**	**105,780**
30.250-20	187	150,163 (Vimont et al. 2008)	**150,163**	150,138
30.250-21	187	149,958 (Angelelli et al. 2010, Vimont et al. 2008)	**149,958**	**149,958**
30.250-22	187	153,007 (Angelelli et al. 2010, Vimont et al. 2008)	**153,007**	**153,007**
30.250-23	186	153,234 (Angelelli et al. 2010, Vimont et al. 2008)	**153,234**	**153,234**
30.250-24	186	150,287 (Angelelli et al. 2010, Vimont et al. 2008)	**150,287**	**150,287**
30.250-25	186	148,574 (Angelelli et al. 2010, Vimont et al. 2008)	**148,574**	**148,574**
30.250-26	186	147,477 (Angelelli et al. 2010, Vimont et al. 2008)	**147,477**	**147,477**
30.250-27	186	152,912 (Angelelli et al. 2010, Vimont et al. 2008)	**152,912**	**152,912**
30.250-28	186	149,570 (Angelelli et al. 2010, Vimont et al. 2008)	**149,570**	**149,570**
30.250-29	187	149,668 (Angelelli et al. 2010)	**149,668**	**149,668**

and, consequently, the size of the largest subproblem dealt with by CORAL are highly correlated with the number of constraints. Fortunately, the percentage of nonsolved problems also increases with number of constraints. The limited number of improvements indicates that the first subproblem solved at step 6 of the initialization phase usually provides a very good quality solution (in the smallest instances, 5.100, this was already the optimal solution). This confirms the relevance of using absolute reduced cost values as an efficiency measure. We take, as an example, the instance 5.500-00. Figure 4 shows the graph of the computational time required to solve each of the 136 subproblems required in step 3 of the iterative phase for such an instance, where the subproblems are numbered in order of creation and solution. The time required to solve a subproblem tends to decrease as the subproblem size increases. This is indeed a common feature shown by CORAL in all solved instances.

Table 10 **Subproblems and Restricted Core Problems: Total Values**

	Subproblems			Restricted core problems		
Problem	No.	Imp.	Max size (%)	No.	Imp.	Nonsolved (%)
5.100-00	37	0	67.0	164	0	20.73
5.100-10	48	0	78.0	90	0	35.56
5.100-20	35	0	65.0	121	0	30.58
5.250-00	80	1	44.0	1,043	3	27.71
5.250-10	91	3	48.4	8,768	4	24.05
5.250-20	92	0	48.8	4,094	0	28.36
5.500-00	136	1	33.2	84,313	1	24.82
5.500-10	134	3	32.8	60,002	5	26.66
5.500-20	117	0	29.4	7,981	0	33.77
10.100-00	67	1	97.0	10,624	1	33.98
10.100-10	68	0	98.0	7,633	0	44.31
10.100-20	45	0	75.0	229	0	33.62
10.250-00	197	1	90.8	1,673,685	1	43.11
10.250-10	191	1	88.4	1,732,519	2	39.35
10.250-20	147	1	70.8	717,537	4	43.98
30.100-00	69	0	99.0	63,437	0	56.97
30.100-10	69	2	99.0	691,953	2	55.33
30.100-20	69	2	99.0	29,202	2	58.14

Table 11 provides statistics on the branch-and-bound routine. Columns provide the minimum, average, and maximum values of the following quantities: the number of variables ("No. of var"), minimum and maximum cardinality ($\bar{k}_{\min}$ and $\bar{k}_{\max}$, respectively) for each solved restricted core problem, and the total number of visited nodes for each call to the branch-and-bound routine ("Total nodes"). Again, the size as well as the minimum and maximum cardinality of solved problems are positively correlated to the number of constraints. Total nodes is a measure of the intensity of the tree search. Each call of the *SearchTree*(k) routine considers a tree with size

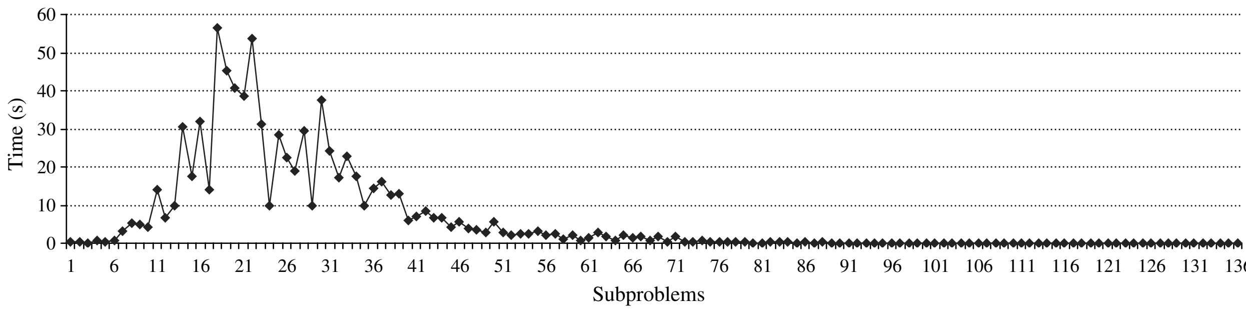

Figure 4 **Instance 5.500-00**
Note. Solution time (in seconds) of subproblems of step 3 of the iterative phase.

$O(2^{n_{\max}})$, with $n_{\max} = 30$. The maximum number of total nodes analyzed by a call to the branch-and-bound routine is on average lower than 2% the maximum size of the tree, reaching almost 6% in the instance 5.500-10, thus showing a correlation with both the number of items and the constraints.

Table 12 shows the effectiveness of the different types of pruning conditions used by the branch-and-bound routine. Recalling that the conditions are applied in sequence, one can notice that, on average, the effectiveness of pruning conditions on value (Condition 1) seems to slightly decrease when the number of constraints increases, whereas the reverse is true for pruning conditions on weight (Condition 4). Reduced costs conditions (Condition 3) are, on average, quite effective when the number of constraints is lower than 10 but reduce for $m = 30$.

Finally, Table 13 gives an idea of the high relevance of the VariableFixing( ) procedure. Statistics are computed over all the subproblems generated through the VariableFixing( ) procedure. More specifically, the first three columns of Table 13 provide the minimum, average and maximum number of restricted core problems out of all solved subproblems MKP$(N, \bar{N}_0, \bar{N}_1)$ in step 3 of the iterative phase. Columns "Recursive calls" provide the same information on the number of recursive calls to routine VariableFixing( ) and, thus, give an idea of the depth of the trees analyzed by the fixing process procedure when solving a subproblem. Finally, columns "No. direct fixing" and "No. temporary fixing" show the minimum, average, and maximum number of variables set to a value by the direct fixing process and by the temporary one, respectively. The larger the size of an instance and the larger the number of constraints, the higher the number of restricted core problems solved and the higher the number of the recursive calls as well as the number of variables set to a value by direct fixing procedure.

Table 11 **Branch-and-Bound Routine: Statistics on Restricted Core Problems**

	No. of var			$\bar{k}_{\min}$			$\bar{k}_{\max}$			Total nodes		
Problem	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max
5.100-00	5	19.61	30	2	7.23	12	3	8.95	15	5	185,038.20	16,325,025
5.100-10	6	21.34	30	3	10.38	16	4	14.11	20	22	165,840.58	6,464,642
5.100-20	6	16.49	30	3	7.95	17	3	9.90	21	10	101,941.77	4,484,307
5.250-00	3	20.33	30	2	10.86	17	2	12.85	20	5	130,318.87	11,099,888
5.250-10	5	20.52	30	2	9.18	17	2	11.68	21	4	224,729.64	29,626,635
5.250-20	5	21.07	30	2	9.51	16	2	12.34	21	3	136,673.48	26,831,141
5.500-00	5	22.63	30	2	8.31	17	2	10.43	20	3	97,581.51	5,056,490
5.500-10	5	21.92	30	2	9.93	19	2	13.11	23	3	251,811.94	59,952,742
5.500-20	5	20.88	30	2	7.44	14	2	10.73	20	3	157,523.46	20,771,097
10.100-00	9	25.01	30	3	9.06	14	3	11.21	18	5	69,994.69	7,048,826
10.100-10	8	24.71	30	2	10.85	18	2	13.45	21	7	200,942.32	23,941,535
10.100-20	13	25.59	30	5	9.45	13	7	12.41	16	76	220,348.72	13,373,414
10.250-00	6	25.84	30	2	9.89	19	2	11.98	21	3	123,330.79	11,226,907
10.250-10	6	25.60	30	1	11.52	22	1	14.15	24	3	246,181.30	22,860,922
10.250-20	6	25.63	30	2	10.79	20	2	12.98	23	5	126,245.63	8,141,023
30.100-00	10	27.56	30	5	12.60	19	5	14.63	21	111	385,595.13	30,647,666
30.100-10	12	27.56	30	4	12.11	20	5	13.97	22	48	270,005.89	18,771,731
30.100-20	11	26.80	30	5	13.68	21	5	15.21	22	69	158,513.98	30,117,940

Table 12 **Number of Pruning Conditions for Each Call to the Branch-and-Bound Routine**

	Value			Total weight			Reduced costs			Weight		
Problem	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max
5.100-00	2	55,436.20	5,697,458	2	34,262.390	3,190,483	0	17,176.080	368,121	0	35,309.23	3,695,431
5.100-10	5	34,580.85	1,455,924	5	28,768.170	1,104,349	0	8,393.635	71,262	0	55,314.46	2,498,351
5.100-20	2	25,655.78	1,238,178	3	19,220.020	860,731	0	6,829.641	62,901	0	19,649.94	1,096,899
5.250-00	1	26,767.99	2,738,642	2	19,002.630	1,419,378	0	19,791.620	581,318	0	23,232.96	4,333,280
5.250-10	1	44,505.66	7,101,220	2	37,789.590	5,883,390	0	37,038.360	679,705	0	43,897.96	9,576,667
5.250-20	1	27,866.56	7,485,585	2	17,163.880	4,027,473	0	28,320.510	836,876	0	25,769.28	9,716,643
5.500-00	1	19,728.73	1,093,764	2	16,118.660	988,883	0	21,446.740	767,278	0	15,704.95	1,434,266
5.500-10	1	47,719.53	15,782,720	2	39,530.030	9,936,420	0	33,333.830	1,547,018	0	60,015.31	20,176,327
5.500-20	1	31,079.71	5,604,787	2	25,372.250	3,268,612	0	37,623.570	1,160,640	0	24,573.76	7,999,911
10.100-00	1	19,001.86	2,577,401	2	9,966.092	673,492	0	8,283.930	375,741	0	13,383.72	2,504,600
10.100-10	2	54,614.85	12,324,196	2	26,186.020	932,918	0	26,678.170	483,140	0	47,179.67	7,056,630
10.100-20	20	59,469	4,214,738	6	25,626.590	1,163,070	0	21,373.220	185,103	14	67,867.33	5,770,015
10.250-00	1	26,603.93	4,149,380	2	17,857.810	801,906	0	16,371.370	828,651	0	30,245.91	4,735,909
10.250-10	1	52,008.82	10,994,888	2	21,907.950	1,290,330	0	36,996.670	1,967,247	0	75,505.06	8,217,611
10.250-20	1	24,872.90	2,232,081	2	19,280.660	1,125,152	0	14,988.040	1,009,714	0	34,789.84	2,752,960
30.100-00	4	76,524.38	13,358,009	2	13,515.750	415,257	0	18,140.740	1,234,451	0	18,140.74	1,234,451
30.100-10	5	43,409.64	6,684,349	0	6,932.549	381,034	0	14,299.110	1,534,800	22	147,327.30	8,568,518
30.100-20	3	22,690.44	10,893,622	0	9,714.733	271,283	0	11,957.650	423,949	17	82,561.83	15,128,822

Table 13 **VariableFixing( ) Routine: Statistics on Solved Subproblems**

	Restricted core problems			Recursive calls			No. direct fixing			No. temporary fixing		
Problem	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max
5.100-00	1	4.43	13	0	0.57	1	0	1.67	6	0	3.48	11
5.100-10	1	1.88	11	0	0.29	1	0	5.29	12	0	2.52	9
5.100-20	1	3.46	12	0	0.46	1	0	3.06	15	0	3.19	10
5.250-00	1	13.04	44	0	2.15	11	0	3.12	22	0	6.58	28
5.250-10	1	96.35	769	0	8.35	28	0	2.06	24	0	5.54	41
5.250-20	1	44.50	188	0	6.32	23	0	2.52	24	0	6.07	37
5.500-00	1	619.95	4,450	0	15.22	45	0	2.26	32	0	4.48	50
5.500-10	1	447.78	2,451	0	14.11	48	0	1.92	30	0	4.94	53
5.500-20	1	68.21	332	0	8.61	32	0	4.36	38	0	7.15	44
10.100-00	1	158.57	678	0	13.60	34	0	3.99	36	0	5.18	42
10.100-10	1	112.25	612	0	11.97	31	0	6.06	35	0	5.37	40
10.100-20	1	5.09	14	0	0.93	4	0	4.62	21	0	2.96	10
10.250-00	1	8,495.86	369,072	0	6.53	48	0	4.84	41	0	3.94	108
10.250-10	1	9,070.78	270,194	0	4.96	43	0	4.16	46	0	3.67	44
10.250-20	1	4,881.20	35,181	0	27.81	78	0	5.09	51	0	4.66	80
30.100-00	1	919.38	3,427	0	24.94	46	0	5.02	37	0	4.19	53
30.100-10	1	10,028.30	34,675	0	33.01	61	0	4.69	36	0	3.75	64
30.100-20	1	423.22	2245	0	20.62	40	0	6.43	48	0	4.75	48

To conclude, the efficiency of our algorithm highly depends on the effectiveness of the inequality (5) on reduced costs. Increasing correlation among items (as for instances with a number of constraints greater than five) implies that the reduced costs of the relaxed optimal solution are closer to each other and thus highly affect the effectiveness of the variable-fixing process. Moreover, the larger the number of constraints of an instance, the larger the number of basic variables. Basic variables have null reduced costs and thus do not contribute to the reduced costs constraint. Nevertheless, a relevant feature of CORAL is that it always requires an almost constant amount of RAM (never larger than 300 MB). On the contrary, we notice that, to optimally solve instances with $n \geq 250$, CPLEX always has high memory requirements. On the most complex instances, this can be a severe drawback, causing CPLEX to run out of memory after some hours of computing time.

4.2. Computational Results: Glover and Kochenberger's (1996) Instances

Glover and Kochenberger's (1996) data set consists of huge correlated instances. We decided to only solve the first seven instances of this class because the last four involve a number of constraints (up to 100) and

Table 14 Computational Results: Glover and Kochenberger's (1996) Instances

Problem	(n, m)	Best known	CORAL value	#	Time (s)	tB (s)
mk_gk01	(100, 15)	3,766	**3,766**	52	942.81	2.65
mk_gk02	(100, 25)	3,958	*3,958*	50	18,000	570.48
mk_gk03	(150, 25)	5,656	5,655	78	18,000	17,441.87
mk_gk04	(150, 50)	5,767	*5,767*	77	18,000	17,243.56
mk_gk05	(200, 25)	7,560	7,559	103	18,000	14,365.50
mk_gk06	(200, 50)	7,677	7,672	101	18,000	17,800.00
mk_gk07	(500, 25)	19,220	19,214	259	18,000	386.84

variables (up to 2,500) too large for an exact method. The size of the first seven instances is challenging for an exact method but is of the same order of magnitude with the hardest Chu and Beasley's (1998) instances.

Results are shown in Table 14. A time limit of five hours was assigned to CORAL. The initial size of the core set (parameter C) was set equal to the maximum value between 30 and the number of constraints. The maximum size for the restricted core problems (parameter $n_{\max}$) was set to 30 for all the tested instances.

The columns in Table 14 provide the instance name, its size as the number of items n and constraints m, the best-known value (obtained by the hybrid heuristic proposed by Vasquez and Hao 2001), the value found by CORAL and the cardinality of its solution (#), its computational time in seconds (Time), and its time to best (the time to find the best integer feasible solution) in seconds (tB). It is worth noting that to get the best-known results, the authors used up to three days (Vasquez and Hao 2001). Values reported in bold mean that an optimal solution was found, whereas figures are in italics if the best-known value was reached. When CORAL did not find the best-known solution, it found a solution that is very close to it. The results show that the number of constraints in the instance seems to be decisive based on the CPU time. In instances with 50 constraints, most of the time was spent solving the first subproblem in the initial phase. More precisely, in instance mk_gk04 the solution of the initial phase required more than 12,000 seconds, whereas in instance mk_gk06 all assigned time is used for such a phase.

5. Conclusions

In this paper, we propose a new exact approach called CORAL for the solution of the MKP. The method is based on the solution of subproblems limited to subsets of variables and is built around some key features as an attempt to quickly find good-quality lower bounds to speed up the search and to quickly fix as many variables as possible to their provably optimal value.

In experiments, CORAL has proven to be a valuable method for instances with a large number of items and a limited number of constraints. In some specific instances, it may require more time with respect to other approaches proposed in the literature, but its average performance is always better, and its behavior is more stable. With respect to a strong commercial software such as CPLEX 10.0, CORAL is always the most efficient method except for instances with a very small number of items (e.g., classes of instances 10.100 and 30.100), where CPLEX performs extremely well.

Acknowledgments

The first author is the corresponding author. The authors thank two anonymous reviewers for fruitful suggestions that helped improve a previous version of this paper.

References

Angelelli, E., R. Mansini, M. G. Speranza. 2010. Kernel search: A general heuristic for the multi-dimensional knapsack problem. *Comput. Oper. Res.* **37**(11) 2017–2026.

Balas, E., E. Zemel. 1980. An algorithm for large zero-one knapsack problems. *Oper. Res.* **28**(5) 1130–1154.

Boussier, S., M. Vasquez, Y. Vimont, S. Hanafi and P. Michelon. 2008. Solving the 0-1 multidimensional knapsack problem with resolution search. *VI ALIO/EURO Workshop Appl. Combin. Optim., Buenos Aires*. Computing Research Repository Article abs/0905.0848.

Chu, P. C., J. E. Beasley. 1998. A genetic algorithm for the multidimensional knapsack problem. *J. Heuristics* **4**(1) 63–86.

Chvátal, V. 1997. Resolution search. *Discrete Appl. Math.* **73**(1) 81–99.

Fayard, D., G. Plateau. 1982. An algorithm for the solution of the 0-1 knapsack problem. *Computing* **28** 269–287.

Fréville, A., G. Plateau. 1994. An efficient preprocessing procedure for the multidimensional 0-1 knapsack problem. *Discrete Appl. Math.* **49**(1–3) 189–212.

Gavish, B., H. Pirkul. 1985. Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality. *Math. Programming* **31**(1) 78–105.

Glover, F., G. A. Kochenberger. 1996. Critical event tabu search for multidimensional knapsack problems. I. H. Osman, J. P. Kelly, eds. *Meta-Heuristics: Theory and Applications*. Kluwer Academic Publishers, Boston, 407–427.

Hanafi, S., C. Wilbaut. 2011. Improved convergent heuristics for the 0-1 multidimensional knapsack problem. *Ann. Oper. Res.* **183**(1) 125–142.

Huston, S., J. Puchinger, P. J. Stuckey. 2008. The core concept for 0/1 integer programming. J. Harland, P. Manyem, eds. *Proc. 14th Comput.: Australasian Theory Sympos. (CRPIT 77)*, Australian Computer Society, Sydney, NSW, 39–47.

Kaparis, K., A. N. Letchford. 2008. Local and global lifted cover inequalities for the 0-1 multidimensional knapsack problem. *Eur. J. Oper. Res.* **186**(1) 91–103.

Kellerer, H., U. Pferschy, D. Pisinger. 2004. *Knapsack Problems*. Springer, Berlin.

Mansini, R., M. G. Speranza. 2002a. A multidimensional knapsack model for asset-backed securitization. *J. Oper. Res. Soc.* **53**(8) 822–832.

Mansini, R., M. G. Speranza. 2002b. Multi-unit combinatorial auctions: An exact approach. *Proc. 5th Internat. Conf. Electronic Commerce Res.* (*ICECR-5*), *Montréal*. CD-ROM. HEC, University of Montréal, Montréal.

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

Martello, S., P. Toth. 2003. An exact algorithm for the two-constraint 0-1 knapsack problem. *Oper. Res.* **51**(5) 826–835.

Oliva, C., P. Michelon, C. Artigues. 2001. Constraint and linear programming: Using reduced costs for solving the zero/one multiple knapsack problem. *Proc. Workshop Cooperative Solvers Constraint Programming (CoSolv 01), Paphos, Cyprus*, 87–98.

Pisinger, D. 1999. Core problems in knapsack algorithms. *Oper. Res.* **47**(4) 570–575.

Puchinger, J., G. R. Raidl, U. Pferschy. 2010. The multidimensional knapsack problem: Structure and algorithms. *INFORMS J. Comput.* **22**(2) 250–265.

Shih, W. 1979. A branch and bound method for the multiconstraint zero-one knapsack problem. *J. Oper. Res. Soc.* **30**(4) 369–378.

Vasquez, M., J. K. Hao. 2001. An hybrid approach for the 0-1 multidimensional knapsack problem. *Proc. 17th Internat. Joint Conf. Artificial Intelligence (IJCAI-01)*, Morgan Kaufmann, San Francisco, 328–333.

Vasquez, M., Y. Vimont. 2005. Improved results on the 0-1 multidimensional knapsack problem. *Eur. J. Oper. Res.* **165**(1) 70–81.

Vimont, Y., S. Boussier, M. Vasquez. 2008. Reduced costs propagation in an efficient implicit enumeration for the 0-1 multidimensional knapsack problem. *J. Combin. Optim.* **15**(2) 165–178.

Wilbaut, C., S. Hanafi. 2009. New convergent heuristics for 0-1 mixed integer programming. *Eur. J. Oper. Res.* **195**(1) 62–74.