Les Cahiers du GERAD ISSN: 0711–2440

Variable Neighbourhood Decomposition Search with Pseudo-Cuts for Multidimensional Knapsack Problem

S. Hanafi, J. Lazić, N. Mladenović,
C. Wilbaut, I. Crévits

G–2009–87

December 2009

Variable Neighbourhood Decomposition Search with Pseudo-Cuts for Multidimensional Knapsack Problem

Saïd Hanafi [a,b,c]
Jasmina Lazić [d,e]
Nenad Mladenović [d,e,f]
Christophe Wilbaut [a,b,c]
Igor Crévits [a,b,c]

[a] *Université Lille Nord de France, F-59000 Lille, France*
[b] *UVHC, LAMIH, F-59313 Valenciennes, France*
[c] *CNRS, UMR 8530, F-59313 Valenciennes, France*
[d] *Brunel University, West London UB8 3PH, UK*
[e] *Mathematical Institute, Serbian Academy of Sciences and Arts, Kneza Mihaila 36, 11000 Belgrade, Serbia*
[f] *and GERAD*

December 2009

Les Cahiers du GERAD

G–2009–87

Abstract

In this paper we propose new matheuristics for solving multidimensional knapsack problem. They are based on the variable neighbourhood decomposition search (VNDS) principle. The set of neighbourhoods is generated by exploiting information obtained from a series of relaxations. In each iteration, we add pseudo-cuts to the problem in order to produce a sequence of not only lower, but also upper bounds of the problem, so that integrality gap is reduced. General-purpose CPLEX MIP solver is used as a black box for solving subproblems generated during the search process. With this approach, we have managed to obtain results comparable with the current state-of-the-art heuristics on the set of large scale multi-dimensional knapsack problem instances. Moreover, we have reached a few new lower bound values for some of the test instances.

Key Words: 0-1 Mixed Integer Programming, Multidimensional Knapsack Problem, Matheuristics, Variable Neighbourhood Search.

Résumé

Dans ce papier nous proposons de nouvelles métaheuristiques pour résoudre le problème du sac à dos multidimensionnel. Elles sont basées sur le principe de la recherche à voisinage variable avec décomposition. L'ensemble des voisinages est généré en exploitant l'information obtenue à partir d'une série de relaxations. À chaque itération, nous ajoutons des pseudo-coupes au problème de façon à produire une séquence de bornes inférieures et supérieures pour le problème et réduire l'écart entre ces bornes. Le solveur CPLEX est utilisé comme boîte noire pour résoudre les sous-problèmes générés durant le processus de recherche. Avec cette approche nous avons obtenu des résultats comparables à ceux des heuristiques efficaces récentes sur un ensemble d'instances de grande taille du problème du sac à dos multidimensionnel. De plus, nous avons atteint quelques nouvelles valeurs de borne inférieure pour certaines des instances tests.

1 Introduction

Multidimensional Knapsack Problem (**MKP**) is a resource allocation problem which can be formulated as follows:

$$(P) \quad \left[\begin{array}{lll} \max \sum_{j=1}^{n} c_j x_j & & \\ \text{s.t.} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i & \forall i \in M = \{1, 2, \ldots, m\} \\ \quad x_j \in \{0, 1\} & \forall j \in N = \{1, 2, \ldots, n\} \end{array} \right.$$

Here, n is the number of items, m is the number of knapsack constraints. The right hand side $b_i\,(i \in M)$ represents capacity of knapsack, $A = [a_{ij}]$ is the weights matrix, whose element a_{ij} represents the resource consumption of the item $j \in N$ in the knapsack $i \in M$, and $c_j\,(j \in N)$ is the profit income for the item $j \in N$. The optimal objective function value of problem (P) is denoted as $\nu(P)$:

$$\nu(P) = \{\max \sum_{j=1}^{n} c_j x_j \mid \sum_{j=1}^{n} a_{ij} x_j \leq b_i,\ \forall i \in M, x_j \in \{0, 1\},\ \forall j \in N\}.$$

Being a special case of the 0-1 Mixed Integer Programming (0-1 **MIP**) problem:

$$(0\text{-}1\ MIP) \qquad \max\{c^T x \mid x \in X\}, \tag{1}$$

where $X = \{x \in \mathbb{R}^n \mid Ax \leq b, x_j \geq 0 \text{ for } j = 1, \ldots, n, x_j \in \{0, 1\} \text{ for } j = 1, \ldots, p \leq n\}$, MKP is often used as a benchmark model for testing general purpose combinatorial optimisation methods.

Wide range of practical problems in business, engineering and science, can be modeled as a MKP problems. They include capital budgeting problem, cargo loading, allocating processors in a huge distributed computer system, cutting stock problem, delivery of groceries in vehicles with multiple compartments and many more. Since MKP is known to be NP-hard [13], there were numerous contributions over several decades to the development of both exact (mainly for the case $m = 1$, see, for instance, [25, 31, 32], and for $m > 1$, see, for instance [2, 11, 14]) and heuristic (for example [3, 18, 39]) solution methods for MKP. For a complete review of these developments and applications of MKP, reader is referred to [9, 41].

Mathematical programming formulation of MKP is especially convenient for the application of some general purpose solver. However, due to the complexity of the problem, sometimes it is not possible to obtain an optimal solution in this way. This is why huge variety of problem specific heuristics has been tailored, their drawback being that they cannot be applied to a general class of problems. An approach for generating and exploiting small sub-problems was suggested in [15], based on selection of consistent variables, depending on how frequently they attain particular values in good solutions and on how much disruption they would cause to these solutions if changed. More recently, a variety of neighbourhood search heuristics for solving optimization problems have emerged, such as Variable Neighbourhood Search (**VNS**) proposed in [27], Large Neighbourhood Search (**LNS**) introduced in [33] and the large-scale neighbourhood search in [1]. In 2005, Glover proposed an adaptive memory projection (AMP) method for pure and mixed integer programming [16], which combines the principle of projection techniques with the adaptive memory processes of tabu search to set some explicit or implicit variables to some particular values. This philosophy gives a useful basis for unifying and extending a number of other procedures: LNS, local branching (**LB**) proposed in [8], the relaxation induced neighbourhood search (**RINS**) proposed in [4], VNS branching [21], or the global tabu search intensification using dynamic programming (**TS-DP**) [40]. LNS and RINS have been applied successfully to solve large-scale mixed integer programming problems. TS-DP is a hybrid method, combining adaptive memory and sparse dynamic programming to explore the search space, in which a move evaluation involves solving a reduced problem through dynamic programming at each iteration. Following the ideas of LB and RINS, another method for solving mixed integer programming problems was proposed in [24]. It is based on the principles of Variable Neighbourhood Decomposition Search (**VNDS**) [20]. This method uses the solution of the linear relaxation of the initial problem to define sub-problems to be solved within the VNDS framework. In [34], a convergent algorithm for pure 0-1 integer programming was proposed. It solves a series of small sub-problems generated by exploiting information obtained through a series of linear programming relaxations. Hanafi and Wilbaut have proposed several enhanced versions of the Soyster's exact algorithm (see [19, 39]). In further text, we refer to this basic algorithm as the Linear Programming-based Algorithm (**LPA**).

In this paper we propose new heuristics for solving 0-1 MIP, which dynamically improve lower and upper bounds on the optimal value within VNDS. Different heuristics are derived by choosing a particular strategy of updating lower and upper bounds, and thus defining different schemes for generating a series of sub-problems. We also propose a two-level decomposition scheme, in which sub-problems derived using one criterion are further divided into subproblems according to another criterion. The proposed heuristics have been tested and validated on the MKP. The results obtained on two sets of available and correlated instances show that our approach is efficient and effective:

- our proposed algorithms are comparable with the state-of-the-art heuristics
- a few new best known lower bound values are obtained.

This paper is organised as follows. In Section 2, we give a brief overview of the LPA [34] and VNDS algorithms [24], since they are directly related to the new heuristics proposed in this paper. We also present needed mathematical notations and definitions. In Section 3, we present the new heuristics based on the mixed integer and linear programming relaxations of the problem and VNDS principle. Next, in Section 4, computational results are presented and discussed in an effort to assess and analyse the performance of the proposed algorithms. In Section 5, some final outlines and conclusions are provided.

2 Related Work

The new heuristics we propose later in this paper are mainly based on the further development of the ideas of LPA [34] and basic VNDS [20]. In this section we provide description of these two algorithms and needed mathematical notation.

The fixation of variables is essential for both LPA and VNDS and therefore needs to be formulated more precisely. This is why we introduce the notion of *reduced problem*. Given an arbitrary binary solution x^0 and an arbitrary subset of variables $J \subseteq N$, the problem reduced from original problem P and associated with x^0 and J can be defined as:

$$P(x^0, J) \quad \left[\begin{array}{lll} \max & c^T x & \\ \text{s.t.} & Ax \leq b & \\ & x_j = x_j^0 & \forall j \in J \\ & x_j \in \{0,1\} & \forall j \in N \end{array} \right.$$

Obviously, the reduced problem is derived from the original one by setting variables with indices in J at values of x^0. We further define the sub-vector associated with the set of indices J and solution x^0 as $x^0(J) = (x_j^0)_{j \in J}$, the set of indices of variables with integer values as $B(x) = \{j \in N \mid x_j \in \{0,1\}\}$, and the set of indices of variables with value $v \in \{0,1\}$ as $B^v(x) = \{j \in N \mid x_j = v\}$. We will also use the short form notation $P(x^0)$ for the reduced problem $P(x^0, B(x^0))$. Apparently, $P(x^0) = P$ if $x^0 \in]0,1[^n$. The LP-relaxation of problem P is denoted as LP(P), i.e. :

$$\text{LP}(P) \quad \left[\begin{array}{lll} \max & c^{\mathrm{T}} x & \\ \text{s.t.} & Ax \leq b & \\ & x_j \in [0,1] & \forall j \in N \end{array} \right.$$

If C is a set of constraints, we will denote with $(P \mid C)$ the problem obtained by adding all constraints in C to the problem P. Let x and y be two arbitrary binary solutions of the problem P, the distance between x and y is then defined as $\delta(x,y) = \sum_{j \in N} \mid x_j - y_j \mid$. If $J \subseteq N$, then we define partial distance between x and y, relative to J, as $\delta(J,x,y) = \sum_{j \in J} \mid x_j - y_j \mid$ (obviously, $\delta(N,x,y) = \delta(x,y)$). More generally, let $\bar{x}$ be an optimal solution of the LP relaxation LP(P) (not necessarily MIP feasible), and $J \subseteq B(\bar{x})$ an arbitrary subset of indices, the partial distance $\delta(J,x,\overline{x})$ can be linearized as follows:

$$\delta(J,x,\overline{x}) = \sum_{j \in J} x_j(1-\overline{x}_j) + \overline{x}_j(1-x_j).$$

Now we can also introduce the following subproblem notation for $k \in \mathbb{N} \cup \{0\}$:

$$P(k, x^0, J) \quad \left[\begin{array}{lll} \max & c^T x & \\ \text{s.t.} & Ax \leq b & \\ & x_j = x_j^0 & \forall j \in J \\ & \delta(J, x, x^0) \leq k & \\ & x_j \in \{0,1\} & \forall j \in N \end{array} \right.$$

Let X be the solution space of the problem P considered. The neighbourhood structures $\{\mathcal{N}_k \mid k = k_{min}, \ldots, k_{max}\}$, $1 \leq k_{min} \leq k_{max} \leq \mid N \mid$, can be defined knowing the distance $\delta(N, x, y)$ between any two solutions $x, y \in X$. The set of all solutions in the kth neighbourhood of $x \in X$ is denoted as $\mathcal{N}_k(x)$, where

$$\mathcal{N}_k(x) = \{y \in X \mid \delta(N, x, y) \ \leq \ k\}.$$

From the definition of $\mathcal{N}_k(x)$, it follows that $\mathcal{N}_k(x) \subset \mathcal{N}_{k+1}(x)$, for any $k \in \{k_{min}, k_{min}+1, \ldots, k_{max}-1\}$, since $\delta(N, x, y) \ \leq \ k$ implies $\delta(N, x, y) \ \leq \ k+1$. It is trivial that, if we completely explore neighbourhood $\mathcal{N}_{k+1}(x)$, it is not necessary to explore neighbourhood $\mathcal{N}_k(x)$.

2.1 Linear Programming Based Algorithm

The LPA consists in generating two sequences of upper and lower bounds until justifying the completion of an optimal solution of the problem. This is achieved by solving exactly a series of sub-problems obtained from a series of linear programming relaxations. In addition, at each iteration LPA reduces the search space by adding a pseudo-cut which guarantees that sub-problems already explored are not revisited. The outline of the LPA is given in Figure 1, in which we consider as input parameters an instance P of the multidimensional knapsack problem and an initial feasible solution x^* of P.

`LPA`(P, x^*)

1 $Q = P$; *proceed* = `true`;
2 **while** (*proceed*) **do**
3 $\quad \overline{x} = \text{LPSOLVE}(\text{LP}(Q))$;
4 $\quad$ **if** $\overline{x} \in \{0,1\}^n$ **then**
5 $\quad\quad x^* = argmax\{c^{\mathrm{T}} x^*, c^{\mathrm{T}} \overline{x}\}$; **break**;
6 $\quad$ **endif**
7 $\quad x^0 = \text{MIPSOLVE}(P(\overline{x}))$;
8 $\quad$ **if** $(c^{\mathrm{T}} x^0 > c^{\mathrm{T}} x^*)$ **then** $x^* = x^0$;
9 $\quad Q = (Q \mid \delta(B(\overline{x}), x, \overline{x}) \geq 1)$;
10 $\quad$ **if** $(\lfloor c^{\mathrm{T}} \overline{x} - c^{\mathrm{T}} x^* \rfloor < 1)$ **then** *proceed* = `false`;
11 **endwhile**
12 **return** x^*.

Figure 1: Linear programming based algorithm.

The LPA was originally proposed for solving pure 0–1 integer programming. Hanafi and Wilbaut proposed in [19, 39] several extensions for both 0–1 integer programming and 0–1 MIP. The validity of pseudo-cuts added within the LPA search process (line 10 in Figure 1) is guaranteed by the Proposition 1. The proof of the proposition can be found in [39].

Proposition 1 *Let P be a given $0-1$ mixed integer programming problem, $\overline{x}$ a solution of* $\text{LP}(P)$ *and x^0 an optimal solution of the reduced problem $P(\overline{x})$. An optimal solution of P is either the solution x^0 or an optimal solution of the problem $(P \mid \delta(B(\overline{x}), x, \overline{x}) \geq 1)$.*

The consequence of the Proposition 1 is Theorem 1, which states the finite convergence of the LPA. The proof of the theorem can be found in [19, 38].

Theorem 1 *The LPA converges to an optimal solution of the input problem or indicates that the problem is infeasible in a finite number of iterations.*

Although LPA is proved to be an exact algorithm, in practice reduced problems $P(\overline{x})$ can be very complex themselves, so LPA is normally used as a heuristic limited by a total number of iterations or a running time (by changing the stopping condition in line 11 in Figure 1).

The basic version of LPA can also be improved by integrating dominance properties (the reader is refered to [19] for more details). In addition, Wilbaut and Hanafi [39] proposed other variants by integrating a MIP-relaxation in which the integer requirement on x is released for a subset of variables of the problem. They developed two new heuristics by combining the LP-relaxation and the MIP-relaxation to define intensification and diversification phases. In the first heuristic the MIP-relaxation is defined from an optimal solution of the LP-relaxation. This algorithm is refered as the Iterative Relaxation based Heuristic (**IRH**). In the second one the two relaxations are used in a parallel way to form the Iterative Independant Relaxation based Heuristic (**IIRH**). We compare our results with those of these algorithms. More details about these heuristics can be found in [39].

2.2 Basic VNDS

Variable neighbourhood decomposition search (VNDS) is a two-level VNS scheme for solving optimisation problems, based upon the decomposition of the problem [20]. Recently, a new variant of VNDS for solving 0-1 MIP problems, called VNDS-MIP, was proposed [24]. This method combines linear programming (LP) solver, MIP solver and VNS based MIP solving method VND-MIP in order to efficiently solve a given 0-1 MIP problem. The outline of the algorithm is given in Figure 2.

`VNDS-MIP`(P, d, x, k_{vnd})
Find an optimal solution $\overline{x}$ of LP(P);
Choose stopping criteria (set *proceed*1=*proceed*2=`true`);
while (*proceed*1) **do**
  $\delta_j = \mid x_j - \overline{x}_j \mid$, $j \in N$; index variables $x_j, j \in N$,
  so that $\delta_1 \leq \delta_2 \leq \ldots \leq \delta_p$, $p = \mid N \mid$
  Set $n_d = \mid \{j \in N \mid \delta_j \neq 0\} \mid$, $k_{step} = [n_d/d]$, $k = p - k_{step}$;
  while (*proceed*2 **and** $k \geq 0$) **do**
    $J_k = \{1, \ldots, k\}$; $x' =$ MIPSOLVE($P(x, J_k)$, x);
    if ($c^{\mathrm{T}}x' > c^{\mathrm{T}}x$) **then**
      $x =$ VND-MIP(P, k_{vnd}, x'); **break**;
    else
      if ($k - k_{step} > p - n_d$) **then** $k_{step} = \max\{[k/2], 1\}$;
      Set $k = k - k_{step}$;
    endif
    Update *proceed*2.
  endwhile
  Update *proceed*1; **if** ($k \leq 0$) **break**;
endwhile
return x.

Figure 2: VNDS for 0-1 MIP.

Input parameters for the VNDS-MIP algorithm are an instance P of the 0-1 MIP problem, parameter d which defines the number of variables to be released in each iteration, initial feasible solution x of P and maximum size k_{vnd} of neighbourhood explored within VND-MIP. In the beginning, LP-relaxation LP(P) is solved to obtain an optimal solution $\overline{x}$ of $LP(P)$. Then, instead of solving just a single reduced problem $P(\overline{x})$, a series of sub-problems $P(x, J_k)$ is solved, where sets $J_k \subseteq N$, $J_k = \{1, 2, \ldots, k\}$, with $J_0 = \emptyset$, are chosen according to distances $\delta(J_k, x, \overline{x})$, until the improvement of the incumbent objective value is reached. This provides higher flexibility since there is possibility of improving the objective function value in much shorter time when solving $P(x, J_k)$, $|J_k| > |B(\overline{x})|$. The maximum number of sub-problems solved by decomposition with respect to current incumbent solution x (lines 6–15 of the pseudo-code in Figure 2) is $d + \log_2(|B(\overline{x})|) < d + \log_2 n$. Since there are 2^n possible values for objective function value, there can be no more than $2^n - 1$ improvements of the objective value. As a consequence, the total number of steps performed by VNDS cannot exceed $2^n(d + \log_2 n)$.[1] If no improvement has occurred by fixing values of variables to those of the current incumbent solution x, then the last sub-problem the algorithm attempts to solve is $P(x, \emptyset) = P$, which is just the original problem. Therefore, the basic algorithm does not guarantee better performance than the general-purpose MIP solver used as a black-box within the algorithm. This means that running the basic VNDS-MIP as an exact algorithm (i.e. with *proceed*1 and *proceed*2 set to logical constant `true` until the end of computation) does not have any theoretical significance. Nevertheless, when used as a heuristic with a time limit, VNDS-MIP has a very good performance (see [24]).

Finally, we give in Figure 3 an algorithmic description of the VND-MIP method we use in this paper (note the differences from the original description provided in [21]). During the VND, the current neighbourhood of the current solution x' is completely explored, and if a better solution x'' is found, then the whole process is iterated, starting from x'' as the current incumbent. During the process we also change the last pseudo-cut according to the status of the solution obtained (optimal, feasible, problem infeasible).

`VND-MIP`(P, k_{max}, x')
$k = 1$;
while $(k \leq k_{max})$ **do**
    $x'' =$ MIPSOLVE$(P(k, x', N), x')$;
    switch `solutionStatus` **do**
        case "`optSolFound`":
            reverse last pseudo-cut into $\delta(N, x', x) \geq k + 1$;
            set $x' = x''$; $k = 1$;
        case "`feasibleSolFound`":
            replace last pseudo-cut with $\delta(N, x', x) \geq 1$;
            set $x' = x''$; $k = 1$;
        case "`provenInfeasible`":
            reverse last pseudo-cut into $\delta(N, x', x) \geq k + 1$;
            set $k = k + 1$;
        case "`noFeasibleSolFound`":
            Go to 20;
    end
end
return x'.

Figure 3: VND for MIPs.

As the previous VNDS algorithm is valid for 0-1 MIP, we can use it directly for solving the MKP. In the next section we propose several new variants with different strategies of updating lower and upper bounds, and with different schemes for generating a sequence of sub-problems to be solved within VNDS. We then compare these new variants with the original one in Section 4.

3 New advanced VNDS based heuristics

The main red drawback of the basic VNDS-MIP is the fact that the search space is not being reduced during the solution process (except for temporarily fixing the values of some variables). This means that the same solution vector may be examined many times, which may affect the efficiency of the solution process. This

[1] Note that the number of possible values of the objective function is limited to 2^n only in case of pure $0-1$ programs. In case of mixed integer programs, there could be infinitely many possible values if objective function contains continuous variables.

naturally leads to the idea of additionally restricting the search space by introducing pseudo cuts, in order to avoid the multiple exploration of the same areas.

3.1 VNDS-MIP with pseudo-cuts

One obvious way to narrow the search space is to add the objective cut $c^{\mathrm{T}}x > c^{\mathrm{T}}x^*$, where x^* is the current incumbent solution, each time the objective function value is improved. This updates the current lower bound on the optimal objective value and reduces the new feasible region to only those solutions which are better (regarding the objective function value) than the current incumbent. In the basic VNDS version, decomposition is always performed with respect to the solution of the linear relaxation LP(P) of the original problem P. This way, the solution process ends as soon as all sub-problems $P(x, J_k)$ are examined. In order to introduce further diversification into the search process, pseudo-cuts $\delta(J, x, \overline{x}) \geq k$, for some subset $J \subseteq B(\overline{x})$ and certain integer $k \geq 1$, are added whenever sub-problems $P(\overline{x}, J)$ are explored, completely or partially, by exact or heuristic approaches. These pseudo-cuts guarantee the change of the LP solution $\overline{x}$ and also updates the current upper bound on the optimal value of the original problem. This way, even if there is no improvement when decomposition is applied with respect to the current LP solution, the search process continues with the updated LP solution. Finally, further restrictions of the solution space can be obtained by keeping all the cuts added within the local search procedure VND-MIP. The pseudo-code of the VNDS-MIP procedure with these modifications, called VNDS-MIP-PC1, is presented in Figure 4.

```
VNDS-MIP-PC1(P, d, x*, k_vnd)
Choose stopping criteria (set proceed1=proceed2=true);
Add objective cut: LB = c^T x*; P = (P | c^T x > LB).
while (proceed1) do
    Find an optimal solution x̄ of LP(P); set UB = ν(LP(P));
    if (B(x̄) = N) break;
     δ_j =| x*_j − x̄_j |; index x_j so that δ_j ≤ δ_{j+1}, j = 1, ..., p − 1
     Set n_d =| {j ∈ N | δ_j ≠ 0} |, k_step = [n_d/d], k = p − k_step;
     while (proceed2 and k ≥ 0) do
         J_k = {1, ..., k}; x' = MIPSOLVE(P(x*, J_k), x*);
         if (c^T x' > c^T x*) then
              Update objective cut: LB = c^T x'; P = (P | c^T x > LB);
              x* = VND-MIP(P, k_vnd, x'); LB = c^T x*; break;
         else
             if (k − k_step > p − n_d) then k_step = max{[k/2], 1};
             Set k = k − k_step;
         endif
         Update proceed2;
     endwhile
     Add pseudo-cut to P : P = (P | δ(B(x̄), x, x̄) ≥ 1);
     Update proceed1;
endwhile
return LB, UB, x*.
```

Figure 4: VNDS-MIP with pseudo-cuts.

As opposed to the basic VNDS-MIP, the number of iterations in the outer loop of VNDS-MIP-PC1 is not limited by the number of possible objective function value improvements, but with the number of all possible LP solutions which contain integer components. There are $\binom{n}{k}2^k$ possible solutions with k integer components, so there are $\sum_{k=1}^{n}\binom{n}{k}2^k = 3^n - 2^n$ possible LP solutions having integer components. Thus, the total number of iterations of VNDS-MIP-PC1 is bounded by $(3^n - 2^n)(d + \log_2 n)$. The optimal objective function value $\nu(P)$ of current problem P is either the optimal value of $(P \mid \delta(B(\overline{x}), x, \overline{x}) \geq 1)$, or the optimal value of $(P \mid \delta(B(\overline{x}), x, \overline{x}) = 0)$ i.e.

$$\nu(P) = \max\{\nu(P \mid \delta(B(\overline{x}), x, \overline{x}) \geq 1),\ \nu(P \mid \delta(B(\overline{x}), x, \overline{x}) = 0)\}.$$

If the improvement of objective value is reached by solving subproblem $P(x^*, J_k)$, but the optimal solution of P is $\nu(P \mid \delta(B(\overline{x}), x, \overline{x}) = 0)$, then the solution process continues by exploring the solution space of $(P \mid \delta(B(\overline{x}), x, \overline{x}) \geq 1)$ and fails to reach the optimum of P. Therefore, VNDS-MIP-PC1 used as an exact method provides the feasible solution of the initial input problem P in a finite number of steps, but does not guarantee the optimality of that solution. One can observe that if sub-problem $P(\bar{x})$ is solved exactly before adding the pseudo-cut $\delta(B(\overline{x}), x, \overline{x}) \geq 1$ in P then the algorithm converges to an optimal solution. Again, in practice, when used as a heuristic with the time limit as a stopping criterion, VNDS-MIP-PC1 has a very good performance (see Section 4).

Improving VNDS-MIP-PC1. To avoid redundance in search space exploration, we introduce another variant based on the following observation. The solution space of $P(x^*, J_\ell)$ is the subset of the solution space of $P(x^*, J_k)$ (with J_k as in line 9 of Figure 4), for $k < \ell, k, \ell \in \mathbb{N}$. This means that, in each iteration of VNDS-MIP-PC1, when exploring the search space of the current subproblem $P(x^*, J_{k-k_{step}})$, the search space of the previous subproblem $P(x^*, J_k)$ gets revisited. In order to avoid this repetition and possibly allow more time for exploration of those areas of $P(x^*, J_{k-k_{step}})$ search space which were not examined before, we can discard the search space of $P(x^*, J_k)$ by adding cut $\delta(J_k, x^*, x) \geq 1$ to the current subproblem. The corresponding pseudo-code of this variant, called `VNDS-MIP-PC2`(P, d, x^*, k_{vnd}), is obtained from `VNDS-MIP-PC1`(P, d, x^*, k_{vnd}) (see Figure 4) by replacing line 9 with the following line $9'$:

$$\begin{array}{ll} 9': & J_k = \{1, \ldots, k\}; x' = MIPSOLVE(P(x^*, J_k) \mid \delta(J_k, x^*, x) \geq 1), x^*); \\ & P = (P \mid \delta(J_k, x^*, x) \geq 1); \end{array}$$

and by dropping line 19 (the pseudo-cut $\delta(B(\overline{x}), x, \overline{x}) \geq 1$ is not used in this heuristic).

The pseudo-cut $\delta(J_k, x^*, x) \geq 1$ does not necessarily change the LP solution but ensures that current subproblem $P(x^*, J_k)$ does not get examined again later in the search process. Since

$$\nu(P) = \max\{\nu(P \mid \delta(J_k, x^*, x) \geq 1),\ \nu(P \mid \delta(J_k, x^*, x) = 0)\},$$

cut $\delta(J_k, x^*, x) \geq 1$ does not discard the original optimal solution from the reduced search space. It is easy to prove that this algorithm finishes in a finite number of steps and either returns an optimal solution x^* of the original problem (if $LB = UB$), or proves the infeasibility of the original problem (if $LB > UB$).

3.2 A second level of decomposition in VNDS

In this section we propose the use of a second level of decomposition in VNDS, in particular for the MKP. The MKP is tackled by decomposing the problem in several subproblems where the number of items to choose is fixed at a given integer value. Formally, let P_h a subproblem obtained from the original problem by adding the hyperplane constraint $e^{\mathrm{T}}x = h$ for $h \in N$ and enriched by objective cut, defined as follows:

$$(P_h) \left[\begin{array}{ll} \max & c^{\mathrm{T}}x \\ \text{s.t.:} & Ax \leq b \\ & c^{\mathrm{T}}x \geq LB + 1 \\ & e^{\mathrm{T}}x = h \\ & x_j \in \{0,1\},\ j \in N \end{array} \right.$$

Solving the 0-1MKP by tackling separately each of the subproblems P_h for $h \in N$ appeared to be an interesting approach [2, 35, 36, 37] particularly because the additional constraint $(e^{\mathrm{T}}x = h)$ provides tighter upper bounds than the classical LP–relaxation. Let h_{min} and h_{max} denote lower and upper bounds of the number of variables with value 1 in an optimal solution of the problem. Then it is obvious that

$\nu(P) = max\{\nu(P_h) \mid h_{min} \leq h \leq h_{max}\}$. Bounds $h_{min} = \lceil \nu(LP_0^-) \rceil$ and $h_{max} = \lfloor \nu(LP_0^+) \rfloor$ can be computed by solving the following two problems:

$$(LP_0^-) \left[\begin{array}{ll} \min & e^{\mathrm{T}}x \\ \text{s.t.:} & Ax \leq b \\ & c^{\mathrm{T}}x \geq LB + 1 \\ & x_j \in [0,1]\,,\ j \in N \end{array} \right. \qquad (LP_0^+) \left[\begin{array}{ll} \max & e^{\mathrm{T}}x \\ \text{s.t.:} & Ax \leq b \\ & c^{\mathrm{T}}x \geq LB + 1 \\ & x_j \in [0,1]\,,\ j \in N \end{array} \right.$$

Exploring hyperplanes in a predefined order. As we previously mentioned, the MKP problem P can be decomposed into several subproblems (P_h), such that $h_{min} \leq h \leq h_{max}$, corresponding to hyperplanes $e^{\mathrm{T}}x = h$. Based on this decomposition, we can derive several versions of the VNDS scheme. In the first variant considered, we define the order of the hyperplanes at the beginning of the algorithm, and then we explore them one by one, in that order. The ordering can be done according to the objective function values of linear programming relaxations LP(P_h), $h \in H = \{h_{min}, \ldots, h_{max}\}$. In each hyperplane, VNDS-MIP-PC2 is applied and if there is no improvement, the next hyperplane is explored. That corresponds to the pseudo-code in Figure 5.

Flexibility in changing hyperplanes. In the second variant we consider the hyperplanes in the same order as in the previous version. However, instead of changing the hyperplane only when the current one is completely explored, we allow the change depending on other conditions (a given running time, a number of iterations without improving the current best solution, ...). Figure 6 provides an algorithmic description of this algorithm, in which *proceed*3 corresponds to the condition for changing the hyperplane.

`VNDS-HYP-FIX`(P, d, x^*, k_{vnd})
1 Solve the LP-relaxation problems LP_0^- and LP_0^+;
  Set $h_{min} = \lceil \nu(LP_0^-) \rceil$ and $h_{max} = \lfloor \nu(LP_0^+) \rfloor$;
2 Sort the set of subproblems $\{P_{h_{min}}, \ldots, P_{h_{max}}\}$ so that
  $\nu(LP(P_h)) \leq \nu(LP(P_{h+1}))$, $h_{min} \leq h < h_{max}$;
3 Find initial integer feasible solution x^*;
4 **for** $(h = h_{min}; h \leq h_{max}; h++)$
5   $x' =$ VNDS-MIP-PC2(P_h, d, x^*, k_{vnd})
6   **if** $(c^{\mathrm{T}}x' > c^{\mathrm{T}}x^*)$ **then** $x^* = x'$;
7 **endfor**
8 **return** x^*.

Figure 5: Two levels of decomposition with hyperplanes ordering.

In this algorithm, we simply increase the value of h by one when the changing condition is satisfied (except when $h = h_{max}$; in that case h is fixed to the first possible value starting from h_{min}). When the best solution is improved we also recompute the values of h_{min} and h_{max}, and we update the set H if needed (line 15). In the same way, if an hyperplane is completely explored (or if it cannot contained an optimal solution) we update set H and we change the value of h (line 8 in Figure 6). In our experiments, the condition *proceed*3 corresponds to a maximum running time fixed according to the size of thre problem (see Section 4 for more details about parameters). It is easy to see that there is no guarantee to find an optimal solution of the input problem with this algorithm.

4 Computational Results

All values presented are obtained using Pentium 4 computer with 3.4GHz processor and 4GB RAM and general purpose MIP solver CPLEX 11.1 [22]. We use C++ programming language to code our algorithms and compile them with g++ and the option -O2.

Test bed. We validate our heuristics on two sets of available and correlated instances of MKP. The first set is composed by 270 instances with $n = 100$, 250 and 500, and $m = 5$, 10, 30. These instances are grouped

```
VNDS-HYP-FLE($P, d, x^*, k_{vnd}$)
Solve the LP-relaxation problems $LP_0^-$ and $LP_0^+$;
set $h_{min} = \lceil \nu(LP_0^-) \rceil$ and $h_{max} = \lfloor \nu(LP_0^+) \rfloor$;
Sort the set of subproblems $\{P_{h_{min}}, \ldots, P_{h_{max}}\}$ so that
$\nu(LP(P_h)) \leq \nu(LP(P_{h+1}))$, $h_{min} \leq h < h_{max}$;
Find initial integer feasible solution $x^*$; $LB = c^{\mathrm{T}} x^*$;
Choose stopping criteria (set proceed1=proceed2=proceed3=true) ;
Set $h = h_{min}$ and $P = P_h$;
while (proceed1)
    Find an optimal solution $\overline{x}$ of LP($P$); $UB = min\{UB, \nu(\mathrm{LP}(P))\}$;
    if ($c^{\mathrm{T}} x^* \geq c^{\mathrm{T}} \overline{x}$ or $B(\overline{x}) = N$) then $H = H - \{h\}$ ;
    Choose the next value $h$ in $H$ and Set $P = P_h$;
    $\delta_j = \mid x_j^* - \overline{x}_j \mid$; index $x_j$ so that $\delta_j \leq \delta_{j+1}$
    Set $n_d = \mid \{j \in N \mid \delta_j \neq 0\} \mid$, $k_{step} = [n_d/d]$, $k = p - k_{step}$;
    while (proceed2 and $k \geq 0$)
        $J_k = \{1, \ldots, k\}$; $x' = \mathrm{MIPSOLVE}(P(x^*, J_k), x^*)$;
        if ($c^{\mathrm{T}} x' > c^{\mathrm{T}} x^*$) then
            Update objective cut: $LB = c^{\mathrm{T}} x'$; $P = (P \mid c^{\mathrm{T}} x > LB)$;
            Recompute $h_{min}$, $h_{max}$ and update $H$, $h$ and $P$ if necessary;
            $x^* = \mathrm{VND\text{-}MIP}(P, k_{vnd}, x')$; $LB = c^{\mathrm{T}} x^*$; break;
        else
            if ($k - k_{step} > p - n_d$) then $k_{step} = \max\{[k/2], 1\}$;
            Set $k = k - k_{step}$;
        endif
        Update proceed3;
        if (proceed3=false) then
            Choose the next value $h$ in $H$ ; set $P = P_h$; proceed3=true; goto 26;
        Update proceed2;
    endwhile
    Add pseudo-cut to $P$ : $P = (P \mid \delta(B(\overline{x}), x, \overline{x}) \geq 1)$;
    Update proceed1.
endwhile
return $LB$, $UB$, $x^*$.
```

Figure 6: Flexibility for changing the hyperplanes.

in the OR-Library, and the larger instances with $n = 500$ are known to be difficult. So we test our methods over the 90 instances with $n = 500$. In particular the optimal solutions of the instances with $m = 30$ are not known, whereas the running time needed to prove the optimality of the solutions for the instances with $m = 10$ is in general very important [2].

The second set of instances is composed by 18 MKP problems generated by Glover & Kochenberger (GK)[17], with number of items n between 100 and 2500, and number of knapsack constraints m between 15 and 100. We select these problems because they are known to be very hard to solve by branch-and-bound technique.

CPLEX parameters. As mentioned earlier, the CPLEX MIP solver is used in each method compared. We choose to set the `CPX_PARAM_MIP_EMPHASIS` to `FEASIBILITY` for the first feasible solution, and then change to the default `BALANCED` option after the first feasible solution has been found.

VNDS Parameters. Several variants of our heuristics use the same parameters. In all the cases we set the value of parameter d to 10, and we set $k_{vnd} = 5$. Furthermore, we allow running time $t_{sub} = t_{vnd} = 300$s for calls to CPLEX MIP solver for subproblems and calls to VND, respectively, for all instances in test bed unless otherwise specified. Finally running time limit is set to 1 hour (3,600 seconds) for each instance.

VNDS-HYP-FLE has another parameter that corresponds to the running time before changing the value of h (see Figure 6). In our experiments this value is fixed at 900s (according to preliminary experiments).

Comparison. Here we provide the detailed results for the basic VNDS-MIP method and the four methods proposed in this paper. In Tables 1–3, we provide the results obtained by all our variants over the instances with $n = 500$ and $m = 5, 10, 30$ respectively. In these tables, in column "Best" we report the optimal value (or the best-known lower bound for $m = 30$). These values were obtained by several recent hybrid methods (see [2, 19, 36, 37, 39]). Then, for each variant of our method, we report the difference between the best value and the value obtained by our heuristic, denoted as "Best-lb", and the corresponding CPU time.

Table 1 shows that our heuristics obtain good results over these instances, except for VNDS-HYP-FIX which reaches only 3 optimal solutions. However, for $m = 5$ and $m = 10$ VNDS-HYP-FIX reaches good quality near-optimal solutions in much shorter time than other methods observed. The results for all variants are very similar, in particular for the average gap less than 0.001% (between 23 and 30 optimal solutions). We also can observe that VNDS-MIP slightly dominates the other variants in term of average running time needed to visit the optimal solutions. These results confirm the potential of VNDS for solving the MKP. Another encouraging point is the good behaviour of VNDS-HYP-FLE (with the visit of 30 optimal solutions over the 30 instances), which confirms the interest of using the hyperplane decomposition, even if the use of this decomposition seems to be sensitive (according to the results of VNDS-HYP-FIX). Finally, the VNDS-MIP-PC2 proves the optimality of the solution obtained for the instance 5.500.1 (the value is referred by a "*" in the table).

Table 2 shows that the behaviour of the heuristics is more different for larger instances. Globally the results confirm the efficiency of VNDS-MIP, even if it visits only 2 optimal solutions. VNDS-MIP-PC1 obtains interesting results with 6 optimal solutions and an average gap less than 0.01%. That illustrates the positive impact of the pseudo-cuts in the VNDS scheme. However the addition of the pseudo-cuts in VNDS-MIP-PC1 and VNDS-MIP-PC2 increases the average running time to reach the best solutions. The results obtained by VNDS-HYP-FIX confirm that this variant converges quickly to good solutions of MKP. However, in general, it soon gets stalled in the local optimum encountered during the search, due to the long computational time needed for exploration of particular hyperplanes. More precisely, since hyperplanes are explored successively, it is possible to explore only the first few hyperplanes within the CPU time allowed. Finally the VNDS-HYP-FLE is less efficient than for the previous instances. The increase of the CPU* can be easily explained by the fact that the hyperplane are explored iteratively. The quality of the lower bound can also be explained by the fact that the "good" hyperplanes can be explored insufficiently. However these results are still encouraging.

The results obtained for the largest instances with $m = 30$ are more difficult to analyse. Indeed, the values reported in Table 3 do not completely confirm the previous results. In particular we can observe that VNDS-MIP-PC2 is the "best" heuristic, if we consider only the average gap. In addition, the associated running time is not significantly greater than the running time of VNDS-MIP. For these very difficult instances it seems that the search space restrictions introduced in this heuristic have a positive effect over the VNDS scheme. Finally, the methods based on the hyperplane decomposition are less efficient for these large instances as expected. That confirms the previous conclusions for $m = 10$ that it is difficult to quickly explore all hyperplanes or perform a good selection of hyperplanes to be explored in an efficient way.

Tables 4–5 are devoted to the results over the OR-Library instances and to the comparison of our heuristics with the variation of the time limit from one hour to two hours. In Table 4 we provide the average results obtained by our heuristics. For each heuristic we report the average gap obtained and the number of optimal solutions (or best-known solutions) visited during the process. The value $\alpha \in \{0.25, 0.5, 0.75\}$ was used to generate the instances according to the procedure in [10], and it corresponds to the correlation degree of the instances. There are ten instances available for each (n, m, α) triplet. The main conclusions of this table can be listed as follows:

- According to the average gap the VNDS-MIP-PC2 (very) slightly dominates the other variants based on VNDS-MIP. Globally it is difficult to distinguish one version based one one-level decomposition from the others.
- The VNDS-HYP-* are clearly dominated in average, but not clearly for $m = 10$ (in particular for VNDS-HYP-FLE and $\alpha > 0.5$).
- The VNDS-HYP-FLE obtains most optimal solutions, especially for $m = 5$ and $m = 10, \alpha = 0.5$.
- All the variants have difficulties in tackling the largest instances with $m = 30$. However, if one hour of running time can be considered as an important value for heuristic approaches, it is necessary to observe that a large part of the optimal values and best-known values for the instances with $m = 10$ and $m = 30$, respectively, were obtained in an important running time (see for instance [2, 37, 39]).

According to the last remark, in Table 5 we report the average results obtained by our heuristics with the running time limit set to two hours. According to the results for instances 5.500 for one hour (see Table 4), the comparison on the running time is not significant. The results provided in this table are interesting since they show that if we increase the running time, then VNDS-MIP-PC1 dominates more clearly the VNDS-MIP heuristic, in particular for the instances with $m = 30$. That confirms the potential of the bounding approach. Globally the results of all the variants are clearly improved, in particular for $m = 30$ and $\alpha < 0.75$. Finally we add a "*" for VNDS-MIP-PC1 when $m = 30$ and $\alpha = 0.25$ since it visits one new best known solution for the instance 30.500-3 with an objective value equal to 115,370.

Tables 6–7 are devoted to the results obtained over the GK set of instances. We first provide in Table 6 the overall results of the heuristics compared to the best-known lower bounds reported in column "Best". These values were obtained by an efficient hybrid tabu search algorithm [35] and by the iterative heuristics proposed by Hanafi and Wilbaut [39]. Due to the very important size of several instances and the important running time needed by the other approaches to obtain the best-known solutions, we use two different values for the running time of our heuristics. We set this value at two hours for the medium size instances, but we allow the process running five hours for the instances MK_GK04, MK_GK06 and MK_GK08 to MK_GK11.

Table 6 demonstrates a global interesting behaviour of our heuristics that visit an important number of best-known solutions, and also two new best solutions. In general the new versions derived from VNDS-MIP with some new modifications converge more quickly to the best solutions. That is particularly the case for the VNDS-MIP-PC1 and VNDS-MIP-PC2. Based on the average gap and average running time values, we may observe that VNDS-MIP-PC1 obtains the best results. The previous conclusions about the hyperplane-based heuristics are still valid: the flexible scheme is more efficient for the medium size instances. However for these instances the results obtained by VNDS-HYP-FIX are more encouraging, with the visit of a new best solution and the convergence to good lower bounds for the larger instances.

To complete the analysis of these results, in Table 7 we compare the results obtained by VNDS-MIP and VNDS-MIP-PC2 with the current best algorithms for these instances. We report in this table the values obtained by Vasquez and Hao [35] in column "V&H", and the lower bounds reported in [39] for the LPA algorithm and its extensions IIRH and IRH. These three algorithms were validated on a similar computer, so we also report the running time needed to reach the best solution by these methods. The hybrid tabu search algorithm of Vasquez and Hao was executed over several distributed computers, and it needs several hours to obtain the best solutions. This table confirms the efficiency of our approach: even if the results are not as good for the GK instances, those obtained over the larger instances clearly demonstrate the potential of this hybridization.

Table 1: Results for the 5.500 instances.

		VNDS-MIP		VNDS-MIP-PC1		VNDS-MIP-PC2		VNDS-HYP-FIX		VNDS-HYP-FLE	
inst.	Best	Best-lb	CPU*	Best-lb	CPU*	Best-lb	CPU*	Best-lb	CPU*	Best-lb	CPU*
5.500-0	120148	0	1524	3	794	5	984	38	120	0	1235
5.500-1	117879	0	138	0	475	0*	2663	35	477	0	590
5.500-2	121131	0	2131	2	80	0	2920	47	167	0	484
5.500-3	120804	0	1703	5	3006	5	2259	17	373	0	3192
5.500-4	122319	0	93	0	7	0	303	0	35	0	323
5.500-5	122024	0	722	0	7	0	43	37	55	0	112
5.500-6	119127	0	580	0	2010	0	2068	20	538	0	322
5.500-7	120568	0	167	0	615	0	1152	32	33	0	535
5.500-8	121586	0	1670	0	2820	11	1109	18	222	0	901
5.500-9	120717	0	868	13	2943	0	984	45	3458	0	3198
5.500-10	218428	0	170	0	316	0	12	6	3211	0	1168
5.500-11	221202	0	704	0	356	0	1508	11	27	0	942
5.500-12	217542	0	1957	6	0	0	2341	0	2	0	1835
5.500-13	223560	0	89	0	10	0	133	26	16	0	301
5.500-14	218966	0	88	0	332	0	310	4	5	0	155
5.500-15	220530	0	1265	0	1562	0	1360	44	3091	0	2075
5.500-16	219989	0	7	0	7	0	24	51	56	0	783
5.500-17	218215	0	1230	0	1875	0	2391	35	812	0	774
5.500-18	216976	0	7	0	25	0	12	24	47	0	244
5.500-19	219719	0	457	0	437	0	833	21	31	0	423
5.500-20	295828	0	5	0	5	0	71	40	135	0	1771
5.500-21	308086	0	2	0	235	0	162	20	226	0	28
5.500-22	299796	0	3	0	176	0	57	29	209	0	80
5.500-23	306480	0	2078	0	20	4	5	11	81	0	35
5.500-24	300342	0	1	0	185	0	26	0	87	0	777
5.500-25	302571	0	1097	6	3076	0	1625	22	40	0	133
5.500-26	301339	0	366	0	1355	0	908	30	213	0	735
5.500-27	306454	0	366	0	1041	0	524	32	11	0	212
5.500-28	302828	0	466	0	716	0	719	61	10	0	549
5.500-29	299910	4	0	4	16	4	67	29	51	0	2209
Avg. CPU*			665		817		919		461		871
Avg. Gap			<0.001		<0.001		<0.001		<0.001		0
#opt			29		23		25		3		30

Table 2: Results for the 10.500 instances.

		VNDS-MIP		VNDS-MIP-PC1		VNDS-MIP-PC2		VNDS-HYP-FIX		VNDS-HYP-FLE	
inst.	Best	Best-lb	CPU*	Best-lb	CPU*	Best-lb	CPU*	Best-lb	CPU*	Best-lb	CPU*
10.500-0	117821	12	326	21	1582	12	419	110	606	12	3044
10.500-1	119249	32	38	32	3547	32	348	109	300	83	2711
10.500-2	119215	4	986	4	1163	4	558	71	300	0	1847
10.500-3	118829	16	613	16	1867	16	2006	54	399	16	1959
10.500-4	116530	21	27	21	1227	50	1828	89	917	37	2748
10.500-5	119504	14	852	25	1469	14	1885	91	300	43	364
10.500-6	119827	62	2411	14	911	55	1812	64	300	50	3494
10.500-7	118344	15	2409	21	2271	32	1299	98	919	11	3485
10.500-8	117815	39	3029	0	1233	36	3338	27	454	39	600
10.500-9	119251	48	1231	33	1531	44	3166	67	351	97	1825
10.500-10	217377	20	2315	0	1257	0	1213	43	2476	0	3399
10.500-11	219077	14	2721	16	2032	11	3025	57	901	14	1281
10.500-12	217847	50	1846	50	2447	50	1404	92	26	0	1814
10.500-13	216868	0	63	0	93	0	363	0	69	0	1528
10.500-14	213873	14	1843	14	689	14	958	65	425	30	1225
10.500-15	215086	24	2579	24	1502	50	1815	11	430	0	1108
10.500-16	217940	33	2476	42	1042	9	1496	52	305	42	663
10.500-17	219990	6	2468	6	2154	6	1475	51	370	41	1954
10.500-18	214382	36	1932	27	3002	30	3385	50	431	31	383
10.500-19	220899	27	877	17	1978	27	1682	63	334	27	3289
10.500-20	304387	37	1990	0	2218	28	582	49	302	34	1841
10.500-21	302379	32	2628	0	2000	0	2113	34	345	38	1650
10.500-22	302417	1	1824	1	2766	8	1372	66	351	17	605
10.500-23	300784	27	2287	41	952	41	1373	46	195	27	3072
10.500-24	304374	0	1258	0	1771	8	2281	33	182	17	3456
10.500-25	301836	40	1285	55	2174	40	3112	0	3509	40	2578
10.500-26	304952	3	663	3	1401	1	393	3	1	7	1501
10.500-27	296478	12	717	12	2742	12	3037	68	335	22	2258
10.500-28	301359	2	2203	6	1044	6	1601	44	358	6	1814
10.500-29	307089	17	1039	17	1044	11	2475	51	102	17	1307
Avg. CPU*			1564		1703		1727		543		1960
Avg. Gap			0.013		0.009		0.013		0.034		0.016
#opt			2		6		3		2		5

Table 3: Results for the 30.500 instances.

		VNDS-MIP		VNDS-MIP-PC1		VNDS-MIP-PC2		VNDS-HYP-FIX		VNDS-HYP-FLE	
inst.	Best	Best-lb	CPU*	Best-lb	CPU*	Best-lb	CPU*	Best-lb	CPU*	Best-lb	CPU*
30.500-0	116056	124	3370	107	1628	47	2543	532	337	192	83
30.500-1	114810	30	2530	61	2945	78	2534	275	909	138	2510
30.500-2	116712	30	2993	51	2721	27	2604	208	912	279	15
30.500-3	115329	63	1796	64	3448	19	2214	135	307	153	7
30.500-4	116525	113	3410	91	3193	70	1955	291	637	119	2795
30.500-5	115741	7	3418	106	2381	7	2864	251	100	161	95
30.500-6	114181	33	542	168	991	159	3240	269	3526	137	1229
30.500-7	114348	98	1881	99	2108	66	712	315	1269	175	132
30.500-8	115419	0	2164	0	1008	0	1652	431	630	310	621
30.500-9	117116	93	3123	12	2448	12	2317	248	605	237	88
30.500-10	218104	71	2520	32	2052	36	3553	175	3389	137	2727
30.500-11	214648	28	1319	57	2927	27	2443	300	3314	213	2423
30.500-12	215978	36	374	36	1808	60	2076	294	3473	122	14
30.500-13	217910	48	3251	48	1987	48	2171	270	3306	218	3
30.500-14	215689	70	497	57	995	57	2892	298	689	103	777
30.500-15	215890	21	2343	23	1346	43	2782	429	17	102	2693
30.500-16	215907	24	2211	24	2505	24	2231	209	1513	129	5
30.500-17	216542	79	47	91	78	97	952	282	921	188	302
30.500-18	217340	3	1456	7	2063	3	660	171	611	111	2768
30.500-19	214739	60	2656	88	2585	52	3142	205	3069	145	3006
30.500-20	301675	19	1468	19	1731	19	1177	224	3303	32	3016
30.500-21	300055	0	2022	7	3348	7	1464	190	304	95	3109
30.500-22	305087	25	1232	49	1661	11	1254	163	3310	148	4
30.500-23	302032	31	3166	17	2810	28	1937	166	932	79	3009
30.500-24	304462	35	3262	49	321	29	2687	158	902	161	302
30.500-25	297012	53	1466	0	3307	26	3044	235	301	170	3309
30.500-26	303364	36	424	22	1825	35	321	285	3	131	2802
30.500-27	307007	8	2744	45	3044	8	3352	347	8	113	3
30.500-28	303199	37	2142	37	1363	37	1845	288	912	135	9
30.500-29	300572	40	2189	56	1259	40	2789	309	305	137	617
Avg. CPU*			2067		2063		2180		1327		1282
Avg. Gap			0.027		0.032		0.023		0.152		0.091
# best			2		2		1		0		0

Table 4: Average results on the OR-Library with 1 hour total running time limit.

		5.500				10.500				30.500				Global
	α	0.25	0.5	0.75	Avg.	0.25	0.5	0.75	Avg.	0.25	0.5	0.75	Avg.	Avg.
Avg. Gap	VNDS-MIP	0	0	<0.001	<0.001	0.022	0.010	0.006	0.013	0.051	0.020	0.009	0.027	0.013
	VNDS-MIP-PC1	0.002	<0.001	<0.001	0.001	0.016	0.009	0.004	0.010	0.066	0.021	0.010	0.032	0.014
	VNDS-MIP-PC2	0.002	0	<0.001	0.001	0.025	0.009	0.005	0.013	0.042	0.021	0.008	0.024	0.012
	VNDS-HYP-FIX	0.002	0.01	0.001	0.014	0.066	0.022	0.013	0.034	0.256	0.122	0.078	0.152	0.067
	VNDS-HYP-FLE	0	0	0	0	0.033	0.009	0.007	0.016	0.164	0.068	0.040	0.091	0.036
					Sub Total				Sub Total				SubTotal	Total
#opt	VNDS-MIP	10	10	9	29	0	1	1	2	1	0	1	2	33
	VNDS-MIP-PC1	6	9	8	23	1	2	3	6	1	0	1	2	31
	VNDS-MIP-PC2	7	10	8	25	0	2	1	3	1	0	0	1	29
	VNDS-HYP-FIX	1	1	1	3	0	1	1	2	0	0	0	0	5
	VNDS-HYP-FLE	10	10	10	30	1	4	0	5	0	0	0	0	35

Table 5: Average results on the OR-Library with 2 hours total running time limit.

		10.500				30.500				Global
		0.25	0.5	0.75	Avg.	0.25	0.5	0.75	Avg.	Avg.
Avg. Gap	VNDS-MIP	0.015	0.004	**0.003**	**0.008**	0.029	0.015	0.009	0.018	0.013
	VNDS-MIP-PC1	**0.014**	0.007	**0.003**	**0.008**	0.020*	**0.012**	0.009	**0.014**	**0.011**
	VNDS-MIP-PC2	0.019	0.009	0.005	0.011	0.034	0.017	**0.006**	0.019	0.015
	VNDS-HYP-FIX	0.055	0.023	0.011	0.029	0.208	0.094	0.052	0.118	0.074
	VNDS-HYP-FLE	0.023	0.007	**0.003**	0.011	0.154	0.061	0.033	0.083	0.047
					Sub Total				Sub Total	Total
#opt	VNDS-MIP	1	3	3	7	1	1	2	4	11
	VNDS-MIP-PC1	2	2	4	**8**	3	2	1	**6**	**14**
	VNDS-MIP-PC2	0	2	2	4	1	0	1	2	6
	VNDS-HYP-FIX	0	1	1	2	0	0	0	0	2
	VNDS-HYP-FLE	1	4	3	**8**	0	0	0	0	8

Table 6: Average results on the GK instances.

Inst.	n	m	Best	VNDS-MIP Best-lb	VNDS-MIP CPU*	VNDS-MIP-PC1 Best-lb	VNDS-MIP-PC1 CPU*	VNDS-MIP-PC2 Best-lb	VNDS-MIP-PC2 CPU*	VNDS-HYP-FIX Best-lb	VNDS-HYP-FIX CPU*	VNDS-HYP-FLE Best-lb	VNDS-HYP-FLE CPU*
GK18	100	25	4528	0	1584	0	51	0	153	2	17	0	80
GK19	100	25	3869	0	22	0	321	0	296	2	2	0	613
GK20	100	25	5180	0	197	0	215	0	15	2	21	0	132
GK21	100	25	3200	0	302	0	63	0	742	2	358	0	713
GK22	100	25	2523	0	61	0	21	0	20	1	354	0	93
GK23	200	15	9235	0	172	0	1202	0	729	1	320	0	552
GK24	500	25	9070	0	2672	0	3021	1	342	4	310	1	309
Mk_GK_01	100	15	3766	0	2	0	5	0	2	0	1	0	11
Mk_GK_02	100	25	3958	0	23	0	32	0	74	0	364	0	714
Mk_GK_03	150	25	5656	0	625	0	1380	0	1307	1	770	0	3749
Mk_GK_04	150	50	5767	0	2458	**-1**	3673	**-1**	6665	0	1926	1	1837
Mk_GK_05	200	25	7560	0	3506	0	3191	**-1**	882	**-1**	4913	0	306
Mk_GK_06	200	50	7678	1	9217	0	9641	1	15688	3	16006	0	8371
Mk_GK_07	500	25	19220	1	815	1	5169	0	3533	2	362	2	7186
Mk_GK_08	500	50	18806	2	5986	0	11658	0	2287	5	5605	7	14379
Mk_GK_09	1500	25	58091	4	3193	2	11148	4	2535	3	1893	5	10852
Mk_GK_10	1500	50	57295	4	13467	4	3787	2	13275	7	13395	8	2888
Mk_GK_11	2500	100	95237	9	11719	8	2581	3	7461	10	7996	14	1141
#best				12		13		11		3		11	
#imp				0		1		2		1		0	

Table 7: Comparison with other methods over the GK instances.

Inst.	V&H	LPA lb	LPA CPU*	IIRH lb	IIRH CPU*	IRH lb	IRH CPU*	VNDS-MIP lb	VNDS-MIP CPU*	VNDS-MIP-PC2 lb	VNDS-MIP-PC2 CPU*
GK18	4528	4528	290	4528	**78**	4528	680	4528	1584	4528	153
GK19	3869	3869	65	3869	71	3869	25	3869	**22**	3869	296
GK20	5180	5180	239	5180	474	5180	365	5180	197	5180	**15**
GK21	3200	3200	**27**	3200	58	3200	245	3200	302	3200	742
GK22	2523	2523	60	2523	90	2523	117	2523	61	2523	**20**
GK23	9235	*9233*	5	9235	**44**	9235	184	9235	172	9235	729
GK24	9070	*9067*	1	*9069*	1168	9070	**2509**	9070	2672	*9069*	342
Mk_GK_01	3766	3766	**1**	3766	5	3766	**1**	3766	2	3766	2
Mk_GK_02	3958	3958	45	3958	50	3958	100	3958	**23**	3958	74
Mk_GK_03	5656	*5655*	457	5656	1924	5656	**32**	5656	625	5656	1307
Mk_GK_04	5767	5767	222	5767	282	5767	472	5767	2458	**5768**	6665
Mk_GK_05	7560	7560	458	7560	1261	7560	636	7560	3506	**7561**	882
Mk_GK_06	7677	7675	1727	**7678**	23993	**7678**	10042	7676	9217	7676	15688
Mk_GK_07	**19220**	19217	287	19219	8374	19219	15769	19219	815	**19220**	3533
Mk_GK_08	**18806**	18803	3998	18805	975	18805	11182	18804	5986	**18806**	2287
Mk_GK_09	58087	58082	396	**58091**	15064	58089	17732	58083	3193	58083	2535
Mk_GK_10	**57295**	57289	1999	57292	6598	57292	6355	57291	13467	57293	13275
Mk_GK_11	**95237**	95228	2260	95229	9463	95229	1921	95228	11719	95234	7461

Statistical Analysis. It is well known that average values are susceptible to outliers, i.e., it is possible that exceptional performance (either very good or very bad) in a few instances influences the overall performance of the algorithm observed. Therefore comparison between the algorithms based only on the averages does not necessarily have to be valid. Furthermore, by observing the average gap values, we can only see that, in general, two-level decomposition methods are dominated by other VNDS-MIP based methods. However, due to the very small differences between gap values, it is hard to say how significant this performance distinction is. Also, it is difficult to single out any of the three proposed one-level decomposition methods. This is why we have carried out statistical tests to verify the significance of differences between the solution quality performances. Since we cannot make any assumptions about the distribution of the experimental results, we apply a nonparametric (distribution-free) Friedman test [12], followed by the Nemenyi [30] post-hoc test, as suggested in [5].

Let $\mathcal{I}$ be a given set of problem instances and $\mathcal{A}$ a given set of algorithms. The Friedman test ranks the performances of algorithms for each data set (in case of equal performance, average ranks are assigned) and tests if the measured average ranks $R_j = \frac{1}{|\mathcal{I}|}\sum_{i=1}^{|\mathcal{I}|} r_i^j$ (r_i^j as the rank of the jth algorithm on the ith data set) are significantly different from the mean rank. The statistic used is:

$$\chi_F^2 = \frac{12\,|\mathcal{I}|}{|\mathcal{A}|\,(|\mathcal{A}|+1)}\left[\sum_{j=1}^{|\mathcal{A}|} R_j^2 - \frac{|\mathcal{A}|\,(|\mathcal{A}|+1)^2}{4}\right],$$

which follows a χ^2 distribution with $|\mathcal{A}|-1$ degrees of freedom. Since this statistic proved to be conservative [23], a more powerful version of the Friedman test was developed [23], with the following statistic:

$$F_F = \frac{(|\mathcal{I}|-1)\chi_F^2}{|\mathcal{I}|\,(|\mathcal{A}|-1)-\chi_F^2},$$

which is distributed according to the Fischer's F-distribution with $|\mathcal{A}|-1$ and $(|\mathcal{A}|-1)(|\mathcal{I}|-1)$ degrees of freedom. For more details, see [5].

The Friedman test is carried out over the entire set of 108 instances (90 instances from the OR library and 18 Glover & Kochenberger instances). Averages over solution quality ranks are provided in Table 8. According to the average ranks, VNDS-HYP-FIX has the worst performance with rank 4.74, followed by the VNDS-HYP-FLE with rank 3.46, whereas all other methods are very similar, with VNDS-MIP-PC2 (the convergent variant) being the best among the others. The value of the F_F statistic for $|\mathcal{A}| = 5$ algorithms and $|\mathcal{I}| = 108$ data sets is 103.16, which is greater than the critical value 4.71 of the F-distribution with $(|\mathcal{A}|-1, (|\mathcal{A}|-1)(|\mathcal{I}|-1)) = (4, 428)$ degrees of freedom at the probability level 0.001. Thus, we can conclude that there is a significant difference between the performances of the algorithms and proceed with the Nemenyi post-hoc test [30], for pairwise comparisons of all the algorithms. According to the Nemenyi test, the performance of two algorithms is significantly different if the corresponding average ranks differ by at least the critical difference:

$$CD = q_\alpha\sqrt{\frac{|\mathcal{A}|\,(|\mathcal{A}|+1)}{6\,|\mathcal{I}|}},$$

where q_α is the critical value at the probability level α that can be obtained from the corresponding statistical table. For $|\mathcal{A}| = 5$ we get $q_{0.05} = 2.728$ (see [5]), so $CD = 0.587$ for $\alpha = 0.05$. From Table 8 we can see that VNDS-HYP-FIX is significantly worse from all the other methods, since its average rank differs more than 0.587 from all the other average ranks. Also, VNDS-HYP-FLE is significantly better than VNDS-HYP-FIX and significantly worse than all the other methods. Apart from that, there is no significant difference between any other two algorithms. Moreover, no more significant differences between the algorithms can be detected even at the probability level 0.1.

It is obvious that the result of the Friedman test above is largely affected by the very high ranks of the two-level decomposition methods, and it is still not clear whether there is any significant difference between the proposed one-level decomposition methods. In order to verify if any significant distinction between these

Table 8: Differences between the average solution quality ranks for all five methods.

Algorithm (Average Rank)	VNDS-MIP (2.25)	VNDS-MIP-PC1 (2.42)	VNDS-MIP-PC2 (2.12)	VNDS-HYP-FIX (4.74)	VNDS-HYP-FLE (3.46)
VNDS-MIP (2.25)	0.00	-	-	-	-
VNDS-MIP-PC1 (2.42)	0.17	0.00	-	-	-
VNDS-MIP-PC2 (2.12)	-0.13	-0.30	0.00	-	-
VNDS-HYP-FIX (4.74)	**2.49**	**2.32**	**2.62**	0.00	-
VNDS-HYP-FLE (3.46)	**1.21**	**1.04**	**1.34**	**-1.28**	0.00

Table 9: Differences between the average solution quality ranks for three one-level decomposition methods.

Algorithm (Average Rank)	VNDS-MIP (2.01)	VNDS-MIP-PC1 (2.14)	VNDS-MIP-PC2 (1.85)
VNDS-MIP (2.01)	0.00	-	-
VNDS-MIP-PC1 (2.14)	0.13	0.00	-
VNDS-MIP-PC2 (1.85)	-0.16	**-0.30**	0.00

three methods can be made, we further perform Friedman test only on these methods, again over the entire set of 108 instances. According to the average ranks (see Table 9), the best choice is the convergent algorithm VNDS-MIP-PC2, with rank 1.85, followed by the basic VNDS-MIP with 2.01, whereas the variant VNDS-MIP-PC1 has the worst performance, having the highest rank 2.14. The value of the F_F statistic for $|\mathcal{A}| = 3$ one-level decomposition algorithms and $|\mathcal{I}| = 108$ data sets is 2.41. The test is able to detect the significant difference between the algorithms at the probability level 0.1, for which the critical value of the F-distribution with $(|\mathcal{A}| - 1, (|\mathcal{A}| - 1)(|\mathcal{I}| - 1)) = (2, 214)$ degrees of freedom is equal to 2.33.

In order to further examine to which extent is VNDS-MIP-PC2 better than the other two methods, we will perform the Bonferroni-Dunn post-hoc test [7]. Bonferroni-Dunn test is normally used when one algorithm of interest (the control algorithm) is compared with all the other algorithms, since in that special case it is more powerful than the Nemenyi test (see [5]). The critical difference used for the Bonferroni-Dunn test is calculated using the same formula $CD = q_\alpha \sqrt{\frac{|\mathcal{A}|(|\mathcal{A}|+1)}{6|\mathcal{I}|}}$ as for the Nemenyi test, but with the different critical values q_α. Again, the performance of an observed algorithm is considered to be significantly different from the performance of the control algorithm, if the corresponding average ranks differ by at least the critical difference CD. For $|\mathcal{A}| = 3$, we have $q_{0.1} = 1.96$ and critical difference $CD = 0.27$. Therefore, from Table 9 we can see that VNDS-MIP-PC2 is significantly better than VNDS-MIP-PC1 at the probability level $\alpha = 0.1$. The post-hoc test is not powerful enough to detect any significant difference between VNDS-MIP-PC2 and VNDS-MIP at this probability level.

Performance profiles. Since small differences in running time can often occur due to the CPU performance, the ranking procedure described above does not necessarily reflect the real observed runtime performance of the algorithms. This is why we use the performance profiling approach for comparing the effectiveness of the algorithms with respect to the computational time (see [6]).

Let $\mathcal{I}$ be a given set of problem instances and $\mathcal{A}$ a given set of algorithms. The *performance ratio* of running time of algorithm $\Lambda \in \mathcal{A}$ on instance $I \in \mathcal{I}$ and the best running time of any algorithm from $\mathcal{A}$ on I is defined as:

$$r_{I,\Lambda} = \frac{t_{I,\Lambda}}{\min\{t_{I,\Lambda} | \Lambda \in \mathcal{A}\}},$$

where $t_{I,\Lambda}$ is the computing time required to solve problem instance I by algorithm Λ. The *performance profile* of an algorithm $\Lambda \in \mathcal{A}$ denotes the cumulative distribution of the performance ratio $r_{I,\Lambda}$:

$$\rho_\Lambda(\tau) = \frac{1}{|\mathcal{I}|}\{I \in \mathcal{I} \mid r_{I,\Lambda} \leq \tau\},\ \tau \in \mathbb{R}.$$

Obviously, $\rho_\Lambda(\tau)$ represents the probability that the performance ratio $r_{I,\Lambda}$ of algorithm Λ is within a factor $\tau \in \mathbb{R}$ of the best possible ratio. The performance profile $\rho_\Lambda : \mathbb{R} \to [0,1]$ of algorithm $\Lambda \in \mathcal{A}$ is a nondecreasing, piecewise constant function. The value $\rho_\Lambda(1)$ is the probability that algorithm Λ solves the most problems in the shortest computational time (compared to all other algorithms). Thus, if we are only interested in the total number of instances which the observed algorithm solves the first, it is sufficient to compare the values $\rho_\Lambda(1)$ for all $\Lambda \in \mathcal{A}$.

Since we have different running time limits for different groups of instances, we decided to employ performance profiling of the proposed algorithms separately for the instances from the OR library (with the running time limit of one hour) and for those GK instances for which the running time limit is set to two hours. The plotting of the performance profiles of all five algorithms for the 90 instances from the OR library is given in Figure 7. We choose to use the logarithmic scale for τ in order to make a clearer distinction between the algorithms for the small values of τ. From Figure 7 it is clear that VNDS-HYP-FIX strongly dominates all other methods for most values of τ. In other words, for most values of τ, VNDS-HYP-FIX has the greatest probability of obtaining the final solution within a factor τ of the running time of the best algorithm. By examining the values $\rho_\Lambda(1)$ in Figure 7 we can conclude that VNDS-HYP-FIX is the fastest algorithm on approximately 48% of instances, basic VNDS-MIP is the fastest on approximately 21% of instances, VNDS-HYP-FLE is the fastest on approximately 19%, VNDS-MIP-PC1 on 8%, and VNDS-MIP-PC2 on 4% of instances. Figure 7 also shows that the basic VNDS-MIP has the best runtime performance among the three one-level decomposition methods.

The performance profiles plot of all five methods for the 12 GK instances with running time limit set to two hours is given in Figure 8. Again, VNDS-HYP-FIX largely dominates all the other methods. However, VNDS-MIP-PC2 dominates the other one-level decomposition for most values of τ, only except the VNDS-MIP for very small values of τ. By observing the values $\rho_\Lambda(1)$ in Figure 8, we can conclude that VNDS-HYP-FIX is the fastest algorithm on approximately 33% of problem instances, VNDS-MIP is the fastest on 25% of instances, VNDS-MIP-PC2 and VNDS-HYP-FLE have the same number of wins and are the fastest on 17% of instances each, whereas VNDS-MIP-PC1 has the lowest number of wins and is the fastest on only 8% of instances. It may be interesting to note that VNDS-HYP-FLE has much worse performance on the GK set: for small values of τ it is only better than VNDS-MIP-PC1, whereas for most larger values of τ it has the lowest probability of obtaining the final solution within the factor τ of the best algorithm.

In summary, we may conclude that VNDS-HYP-FIX is by far the fastest method for obtaining the acceptable near-optimal solutions of MKP instances. Since it is not as good regarding the solution quality, its purpose may be two fold. On one hand, one may opt for VNDS-HYP-FIX for a good feasible solution in a (very) short time. On the other hand, the VNDS-HYP-FIX may be used as a first-stage of another method, so that a solution obtained with VNDS-HYP-FIX in a short time can be further improved using some other techniques. We may also conclude that VNDS-MIP-PC2 is the best choice both regarding the solution quality and the computational time on the GK set. On the set of instances from the OR-library, the good solution quality performance of VNDS-MIP-PC2 has the price of longer running time.

5 Conclusion

Most of discrete and continuous optimisation problems are hard to solve. The idea of combining exact solution methods, which use mathematical programming formulation, and metaheuristics has attracted a lot of attention recently. As a consequence, a new class of methods, called *matheuristics* (or *model-based* heuristics), has been introduced. In this paper we propose matheuristic methods for solving one very well

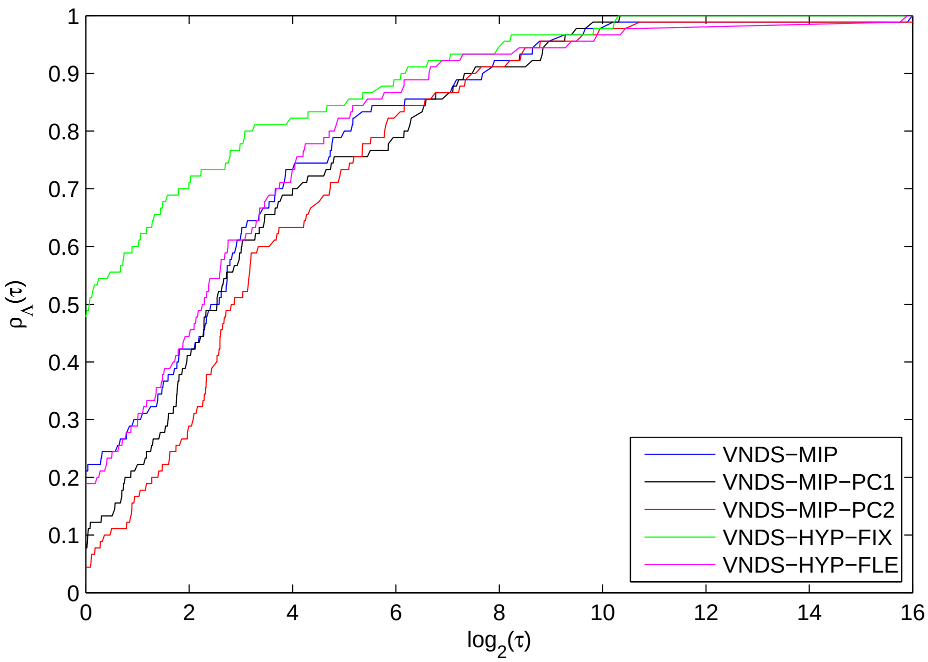

Figure 7: Performance profiles of all 5 algorithms over the OR library data set.

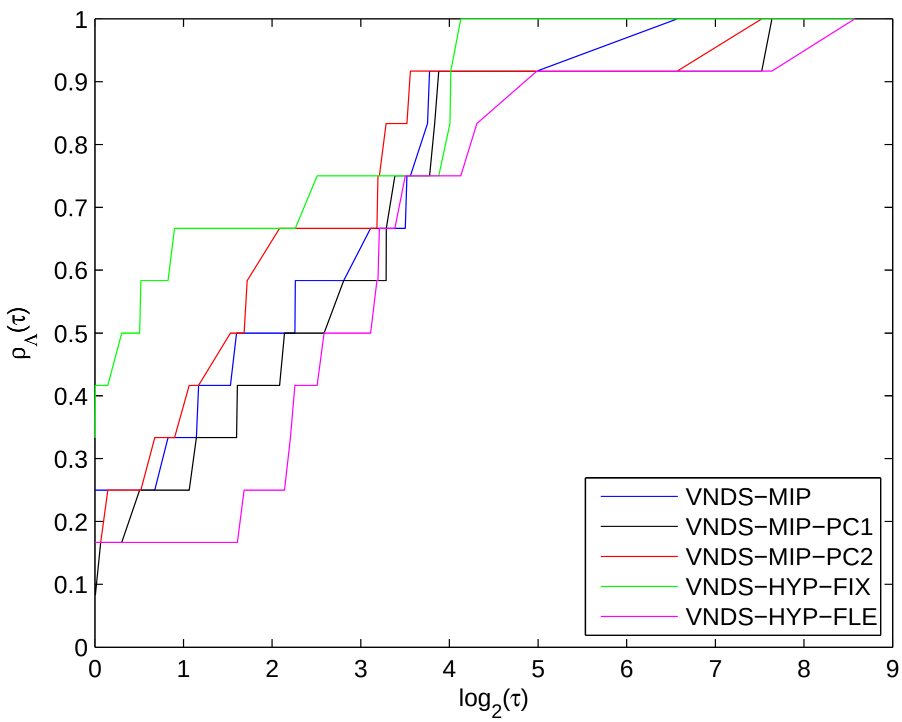

Figure 8: Performance profiles of all 5 algorithms over the GK data set.

known and well studied problem, Multidimensional Knapsack Problem (MKP). We combine solutions of exact MIP solver and variable neighborhood decomposition search (VNDS) metaheuristic.

Variable neighborhood search (VNS) is a metaheuristic which exploits the idea of neighborhood change in search for better solutions. It has several variants that follow this basic idea. One among them is variable neighborhood decomposition search (VNDS). It consists of systematic fixing of certain number of solution attributes and solving the remaining smaller size problems by using the basic VNS. VNDS has already been proposed as a mean for solving general mixed integer programming problems (MIPs) [24]. In [24] variables are ranked in non-decreasing order of the difference between the incumbent solution and the optimal LP relaxation of MIP. Following the VNDS scheme, the upper bound (in the case of minimisation) is updated each time the incumbent is improved. The lower bound remained unchanged. In this paper we propose a few new variants of VNDS for solving MKP, which introduce pseudo-cuts and objective cuts during the execution of the code. In other words, we update both lower and upper bounds in different ways, in order to reduce the integrality gap.

Based on extensive computational analysis performed on benchmark instances from the literature and several statistical tests designed for the comparison purposes, we may conclude that VNDS based matheuristic has a lot of potential for solving MKP. One of our variants, VNDS-MIP-PC2, which is also theoretically shown to converge to an optimal solution, performs better than others in terms of solution quality. Another one, VNDS-HYP-FIX, although not being as effective as others in terms of solution quality, is the fastest in solving very large test instances. Beside the fact that our new matheuristic methods are comparable with other recent algorithms, we were able to find several new best known solutions on benchmark test instances.

Future research may include application of our VNDS based matheuristic methods for solving other hard integer programming problems, such as General Assignment Problem [26]. In addition, our matheuristic methods may be combined with Formulation Space Search (FSS) based approach [28, 29], producing even more efficient algorithm.

References

[1] R.K. Ahuja, Ö. Ergun, J.B. Orlin, A.P. Punnen, A survey of very large-scale neighborhood search techniques, Discrete Applied Mathematics 123 (1–3) (2002) 75–102.

[2] S. Boussier, M. Vasquez, Y. Vimont, S. Hanafi, P. Michelon, A multi-level search strategy for the 0-1 multidimensional knapsack, *Accepted for publication in Discrete Applied Mathematics*, 2010.

[3] P.C. Chu, J.E. Beasley, A genetic algorithm for the multidimensional knapsack problem, Journal of Heuristics 4 (1998) 63–86.

[4] E. Danna, E. Rothberg, C. Le Pape, Exploring relaxation induced neighborhoods to improve mip solutions, Mathematical Programming 102 (1) (2005) 71–90.

[5] J. Demšar, Statistical comparisons of classifiers over multiple data sets, The Journal of Machine Learning Research 7 (2006) 1–30.

[6] E.D. Dolan, J.J. Moré, Benchmarking optimization software with performance profiles, Mathematical Programming (2002) 201–213.

[7] O.J. Dunn, Multiple comparisons among means, Journal of the American Statistical Association (1961) 52–64.

[8] M. Fischetti, A. Lodi, Local branching, Mathematical Programming 98 (2) (2003) 23–47.

[9] A. Fréville, The multidimensional 0–1 knapsack problem: An overview, European Journal of Operational Research 155 (1) (2004) 1–21.

[10] A. Fréville, G. Plateau, An efficient preprocessing procedure for the multidimensional 0–1 knapsack problem, Disccrete Applied Mathematics 49 (1994) 189–212.

[11] A. Fréville, G. Plateau, The 0-1 bidimensional knapsack problem: towards an efficient high-level primitive tool, Journal of Heuristics 2 (1996) 147–167.

[12] M. Friedman, A comparison of alternative tests of significance for the problem of m rankings, The Annals of Mathematical Statistics 11 (1) (1940) 86–92.

[13] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-completeness, WH Freeman San Francisco, 1979.

[14] B. Gavish, H. Pirkul, Efficient algorithms for solving multiconstraint zeroone knapsack problems to optimality, Mathematical Programming 31 (1985) 78–105.

[15] F. Glover, Heuristics for Integer Programming Using Surrogate Constraints, Decision Sciences 8 (1) (1977) 156–166.

[16] F. Glover, Adaptive memory projection methods for integer programming, in: C. Rego, B. Alidaee (Eds.), Metaheuristic Optimization Via Memory and Evolution, Kluwer Academic Publishers, 2005, pp. 425–440.

[17] F. Glover, G.A. Kochenberger, Critical event tabu search for multidimensional Knapsack problems, In: Osman, I., Kelly, J. (Eds.), Meta Heuristics: Theory and Applications (1996) 407–427.

[18] S. Hanafi, A. Fréville, An efficient tabu search approach for the 0–l multidimensional knapsack problem, European Journal of Operational Research 106 (1998) 659–675.

[19] S. Hanafi, C. Wilbaut, Improved convergent heuristics for the 0-1 multidimensional knapsack problem, Annals of Operations Research DOI 10.1007/s10479-009-0546-z.

[20] P. Hansen, N. Mladenović, D. Perez-Britos, Variable Neighborhood Decomposition Search, Journal of Heuristics 7 (4) (2001) 335–350.

[21] P. Hansen, N. Mladenović, D. Urošević, Variable neighborhood search and local branching, Computers & Operations Research 33 (10) (2006) 3034–3045.

[22] ILOG, Cplex 11.1. user's manual (2008).

[23] R.L. Iman, J.M. Davenport, Approximations of the critical region of the Friedman statistic, Communications in Statistics – Theory and Methods 9 (1980) 571–595.

[24] J. Lazić, S. Hanafi, N. Mladenović, D. Urošević, Variable neighbourhood decomposition search for 0–1 mixed integer programs, Computers and Operations Research 37 (6) (2010) 1055–1067.

[25] S. Martello, P. Toth, Upper bounds and algorithms for hard 0–1 knapsack problems, Operations Research 45 (1997) 768–778.

[26] S. Mitrović-Minić, A. Punnen, Local search intensified: Very large-scale variable neighborhood search for the multi-resource generalized assignment problem, Discrete Optimization 6 (2009) 370–377.

[27] N. Mladenović, P. Hansen, Variable neighborhood search, Computers & Operations Research 24 (11) (1997) 1097–1100.

[28] N. Mladenović, F. Plastria, D. Urošević, Reformulation descent applied to circle packing problems, Computers and Operations Research 32 (2005) 2419–2434.

[29] N. Mladenović, F. Plastria, D. Urošević, Formulation space search for circle packing problems, em Lecture Notes in Computer Science 4638 (2007) 212–216.

[30] P. Nemenyi, Distribution-free multiple comparisons, Ph.D. thesis, Princeton. (1963).

[31] D. Pisinger, An expanding-core algorithm for the exact 0–1 knapsack problem, European Journal of Operational Research 87 (1995) 175–187.

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

[33] P. Shaw, Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems, Lecture Notes in Computer Science (1998) 417–431.

[34] A.L. Soyster, B. Lev, W. Slivka, Zero-One Programming with Many Variables and Few Constraints, European Journal of Operational Research 2 (3) (1978) 195–201.

[35] M. Vasquez, J.K. Hao, Une approche hybride pour le sac–à–dos multidimensionnel en variables 0–1, RAIRO Operations Research 35 (2001) 415–438.

[36] M. Vasquez, Y. Vimont, Improved results on the 0–1 multidimensional knapsack problem, European Journal of Operational Research 165 (2005) 70–81.

[37] Y. Vimont, S. Boussier, M. Vasquez, Reduced costs propagation in an efficient implicit enumeration for the 01 multidimensional knapsack problem, Journal of Combinatorial Optimization 15 (2008) 165–178.

[38] C. Wilbaut, Heuristiques hybrides pour la résolution de problèmes en variables 0-1 mixtes, Ph.D. thesis, Université de Valenciennes, Valenciennes, France (2006).

[39] C. Wilbaut, S. Hanafi, New convergent heuristics for 0–1 mixed integer programming, European Journal of Operational Research 195 (2009) 62–74.

[40] C. Wilbaut, S. Hanafi, A. Fréville, S. Balev, Tabu search: global intensification using dynamic programming, Control and Cybernetics 35 (3) (2006) 579–598.

[41] C. Wilbaut, S. Hanafi, S. Salhi, A survey of effective heuristics and their application to a variety of knapsack problems, IMA Journal of Management Mathematics 19 (2008) 227–244.