Optimization Methods and Software
Vol. 00, No. 00, Month 200x, 1–19

RESEARCH ARTICLE

An implementation of exact knapsack separation

Igor Vasilyev[a] *, Maurizio Boccia[b] and Saïd Hanafi[c]

[a] *Institute for System Dynamics and Control Theory Siberian Branch of Russian Academy of Sciences Lermontov Srt., 134, 664033 Irkutsk, Russia. Email: vil@icc.ru*;
[b] *Dipartimento di Ingegneria Università del Sannio Viale Traiano, 82100 Benevento, Italy. E-mail: maurizio.boccia@unisannio.it* ;
[c] *University of Valenciennes LAMIH - UMR CNRS 8530 Le Mont Houy - ISTV2, F-59313 Valenciennes cedex 9, France. E-mail: Said.Hanafi@univ-valenciennes.fr* ;
(*Received 00 Month 200x; in final form 00 Month 200x*)

Cutting planes have been used with great success for solving mixed integer programs. In recent decades, many contributions have led to successive improvements in branch-and-cut methods which incorporate cutting planes in branch and bound algorithm. Using advances that have taken place over the years on 0–1 knapsack problem, we investigate an efficient approach for 0–1 programs with knapsack constraints as local structure. Our approach is based on an efficient implementation of knapsack separation problem which consists of the four phases: preprocessing, row generation, controlling numerical errors and sequential lifting. This approach can be used independently to improve formulations with cutting planes generated or incorporated in branch and cut to solve a problem. We show that this approach allows us to efficiently solve large-scale instances of generalized assignment problem, multilevel generalized assignment problem, capacitated p-median problem and capacitated network location problem to optimality.
Keywords: knapsack problem; cutting plane; exact separation; generalized assignment problem, multilevel generalized assignment problem, capacitated p-median problem, capacitated network location problem.

1. Introduction

The binary integer linear programming problem consists of maximizing or minimizing a linear function, subject to linear constraints and binary choice restrictions on the variables. The binary integer programming problem can be expressed as

$$v_1 = \max_x \{c^T x : \ Ax \leq b, \ x \in \{0,1\}^n\}. \tag{1}$$

The input data are the matrices $c \in \mathbb{Z}^{n\times 1}$, $A \in \mathbb{Z}^{m\times n}$, $b \in \mathbb{Z}^{m\times 1}$, which are supposed to have integer entries. The decision binary variables $x_j \in \{0,1\}$ for $j \in N = \{1,\ldots,n\}$ are of great importance because they regularly occur in many model formulations of real problems in business, engineering, science, transportation etc.

Let

$$X(A,b) = \{x \in \{0,1\}^n : Ax \leq b\}$$

be the set of feasible solutions of (1). The linear programming relaxation of (1) is

*Corresponding author. Email: vil@icc.ru

ISSN: 1055-6788 print/ISSN 1029-4937 online

DOI: 10.1080/03081080xxxxxxxxx
http://www.tandfonline.com

obtained by ignoring the integrality requirements on x, i.e.

$$v_2 = \max_x \{c^T x : \ x \in R(A,b)\}, \tag{2}$$

where

$$R(A,b) = \{x \in [0,1]^n : Ax \leq b\}$$

is its set of feasible solutions. Let $P(A,b)$ be the convex hull of feasible solutions of $X(A,b)$, i.e.

$$P(A,b) = conv(X(A,b)).$$

The polyhedral approach uses the important facts that the set $P(A,b)$ is a polyhedron and the extreme points of $P(A,b)$ are in $X(A,b)$. Thus, (1) can be formulated as the following equivalent linear program

$$v_3 = \max_x \{c^T x : \ x \in P(A,b)\}, \tag{3}$$

i.e. $v_1 = v_3$.

The weakness of the above formulation (3) lies in the difficulty in giving an explicit description of the inequalities defining $P(A,b)$ particularly for NP-hard problems. Moreover, in many cases the number of inequalities to describe $P(A,b)$ is exponential, in the literature there are several approaches which try to find effective ways to construct improved approximations of $P(A,b)$ recursively starting from those defining $R(A,b)$. However, it is not necessary to have an explicit representation of a polyhedron $P(A,b)$ in terms of linear inequalities in order to optimise a linear function over $P(A,b)$. It is enough to be able to solve the separation problem, which is a fundamental tool in polyhedral methods. Using advances on knapsack separation problem that have taken place over the years, in this paper we investigate an efficient approach for 0–1 programs with knapsack constraints as local structure. This approach can be used independently to improve formulations with cutting planes generated or incorporated in the branch-and-cut method to solve the problem.

Let $\widetilde{P}(A,b) = \bigcap_{i=1}^{m} P(A_i,b_i)$, where $P(A_i,b_i) = conv(X(A_i,b_i))$ with

$$X(A_i,b_i) = \{x \in \{0,1\}^n : A_i x \leq b_i\},$$

which define knapsack polytopes. It is trivial that $P(A,b) \subseteq \widetilde{P}(A,b) \subseteq R(A,b)$. So we have

$$v_1 = v_3 \leq v_4 \leq v_3$$

where

$$v_4 = \max_x \{c^T x : \ x \in \widetilde{P}(A,b)\}.$$

For some specially structured problem, like the generalized assignment problem etc., v_4 can be obtained using a dual formulation by the column generation approach or Lagrangian relaxations [15, 16, 32]. But it has been demonstrated that the approach based on an exact knapsack separation algorithm in the cutting plane

methods outperforms the methods used the column generation on the generalized assignment problem and capacitated p-median problem [3, 9, 36]. In this paper we propose a new implementation of this approach, where we try to make it more efficient. In Section 2, we present exact separation problem and its specialization to the knapsack separation. The quality of the resulting cuts focused on tightening the representation is illustrated via several BIP problems in Section 3. Finally, Section 4 discusses conclusions.

2. Exact knapsack separation

The separation problem is one of the basic problems of convex analysis. In our case, for a given point, we should recognize either the point belongs to a problem polytope or find a separation hyperplane which cuts a point from the polytope.

More precisely, let a point $\bar{x} \in \mathbb{R}^n$, and a polytope P be given. We have to prove that $\bar{x} \in P$ or find an inequality $\pi^T x \leq \pi_0$ which is valid for P and such that $\pi^T \bar{x} > \pi_0$. This inequality is called a cutting plane. The separation problem can be reduced to the following optimisation problem

$$v(\bar{x}) = \max_{\pi \in \Pi} \left\{ \min_{x \in P} \left\{ \pi^T \bar{x} - \pi^T x \right\} \right\}, \tag{4}$$

where Π is a convex compact set containing the origin in its interior. If $v(\bar{x}) \leq 0$ then $\bar{x} \in P$, otherwise ${\pi^*}^T x \leq {\pi^*}^T \bar{x}$ is the required cutting plane, where π^* is an optimal solution of (4). Problem (4) is convex and non-differential and it can be transformed to the following linear program:

$$\max_{(\pi, \pi_0)} \left[\bar{x}^T \pi - \pi_0 \right], \tag{5}$$

$$h^T \pi \leq \pi_0 \quad \forall h \in P, \tag{6}$$

$$\pi \in \Pi. \tag{7}$$

The separation problem (5)-(7) for the knapsack polytope was first studied by Boyd [11–14] as a tool for Integer Programming. The recent papers by Fukasawa and Goycoolea [23], Kaparis and Letchford [26] and the dissertations of P. Bonami [10] and D. Espinoza [21] show a renewed interest in this topic. The exact separation problem was also studied for other kinds of problems such that the the knapsack set with a single continues variable [5], the general mixed knapsack set [23], the set covering problem [4], the "local cuts" for the travelling salesman problem [1, 2].

Let us consider the exact separation in an application to the polytope of knapsack problem. Let $X(a, b) = \{x \in \{0, 1\}^n : a^T x \leq b\}$ be the set of feasible solutions of a knapsack problem and $P(a, b) = conv(X(a, b))$ is the corresponding polytope. First of all, the "normalization", which is defined by the set Π, can be specified. It is known for the general cutting plane algorithms [10, 19], that the normalization can play a very important role in the efficiency of generated cuts. For the knapsack polytope this issue has been studied in [3]. It has been shown that the normalization, which provides us the maximum of the ratio between the violation and the right hand side, gives better results in comparison with L_1 and L_∞ norms.

Due to the fact that the trivial inequalities $x_j \geq 0$ are facet-inducing and the remaining facet-inducing inequalities have nonnegative coefficients, such a kind of normalization can be expressed by posing the right-hand side to 1. Thus, the

separation LP (5)-(7) for given $\bar{x}$ for the knapsack polytope becomes:

$$\bar{x}^T\pi^* = \max_{\pi} \left\{ \bar{x}^T\pi : \ x^T\pi \leq 1 \ \ \forall x \in X(a,b) \right\}. \tag{8}$$

If $\bar{x}^T\pi^* \leq 1$, then $\bar{x} \in P(a,b)$; otherwise, $\pi^{*T}x \leq 1$ is a valid inequality separating $\bar{x}$ from $P(a,b)$. There is also a very important property of this LP, that if π^* is a vertex of the feasible set of problem (8), this inequality is facet-inducing. Therefore all our future studies focus only on this normalization.

In practice, problem (8) cannot be written explicitly because it contains a large number of constraints. Our approach to solve it consists of the four following steps, which are detailed in the subsections bellow:

Preprocessing. The fractional knapsack polytope

$$P(a,b,\bar{x}) = \{x \in P(a,b) : \ x_j = \bar{x}_j, \text{ if } \bar{x}_j \in \{0,1\}\}$$

is considered and some simple preprocessing is done.
Row generation. A cutting plane is found by solving the separation LP (8) over the $P(a,b,\bar{x})$ using a *row generation* algorithm.
Numerical errors. After a cutting plane has been generated, it is post-processed to avoid numerical errors.
Sequential lifting. A resulting cut is lifted to the original space with the help of the lifting theorem.

2.1. *Preprocessing*

Without loosing of generality, let us suppose that exactly k first variables are unfixed in the fractional polytope $P(a,b,\bar{x})$, i.e. it can be defined by

$$P(a,b,\bar{x}) = conv(X(a,b,\bar{x})),$$

where

$$X(a,b,\bar{x}) = \left\{ x \in \{0,1\}^k : \sum_{j=1}^{k} a_j x_j \leq \bar{b} \right\}, \ \ \bar{b} = b - \sum_{j \in N : \bar{x}_j = 1} a_j.$$

Our preprocessing procedure is based on the following evident observations:

(1) If $k = 0$ or $\sum\limits_{j=1}^{k} a_j \leq \bar{b}$ then $\bar{x} \in P(a,b,\bar{x})$ and a violated cut does not exist.
(2) Let us consider a subset of items which do not fit the fractional knapsack, i.e. $J^0(\bar{x}) = \{j = \overline{1,k} : a_j > \bar{b}\}$. If $J^0(\bar{x}) \neq \emptyset$ then it immediately follows that

$$\sum_{j \in J^0(\bar{x})} x_j \leq 0$$

is a valid inequality for $P(a,b,\bar{x})$ and it is violated by $\bar{x}$, so it can directly be passed to the sequential lifting.
(3) If $k = 1$, there is only one violated cut $x_1 \leq 0$ which can appear only in case $a_1 > \bar{b}$, i.e. it is processed by the previous observations.

(4) If $k = 2$, there is only one facet-inducing non-trivial inequality

$$x_1 + x_2 \leq 1,$$

which can directly be passed to the sequential lifting in the case of violation.

2.2. *Row generation*

Separation problem (8) is a LP problem, where each row of the constraint matrix is a feasible solution of the knapsack problem. Even for low-dimensional problems, one hardly can enumerate all the feasible points. *Row generation method* is an iterative approach where, at each iteration, a *partial separation problem* (the problem which includes only a subset of the constraints) is considered. It starts with a subset of constraints, which guarantee the boundedness of solution, then other constraints are considered as cutting planes, i.e. they are dynamically added in the case of violation. To check if there is any violated cut, a knapsack problem has to be solved. The main steps of the row generation procedure are summarized below.

Step 0. Choose an initial subset of rows of the constrain matrix of problem (8) $U \subset X(a, b, \bar{x})$. To avoid the unboundness of partial separation problem, one can take unit vectors as the initial set of constraints, i.e. $U = \bigcup_{i=1}^{k} \{e^i\}$.
Step 1. Solve the partial separation problem on the set U:

$$\bar{x}^T\bar{\pi} = \max_{\pi} \left\{ \bar{x}^T\pi : \ x^T\pi \leq 1 \ \ \forall x \in U \right\}. \tag{9}$$

Step 2. Check whether $\bar{\pi}$ satisfies to all the constraints of problem (8). Then find a solution of the knapsack problem:

$$\bar{h} \in \operatorname*{Argmax}_{h} \{\bar{\pi}^T h : h \in X(a, b, \bar{x})\}.$$

Step 3. If $\bar{h}^T\bar{\pi} > 1$, then add $\bar{h}$ to the constraint matrix of problem (9), i.e. $U := U \cup \{\bar{h}\}$, and goto Step 1.
Step 4. If $\bar{h}^T\bar{\pi} \leq 1$, then $\bar{\pi}$ is a solution of separation problem (8) and $\bar{\pi}^T x \leq 1$ is a valid inequality for $P(a, b, \bar{x})$, cutting $\bar{h}$ if $\bar{\pi}^T\bar{h} > 1$.

The row generation procedure requires solving a large number of knapsack problems, so using efficient knapsack algorithms is a key issue. In our computational experiments we used the modification of Pisinger's MINKAP algorithm [34] that combines dynamic programming with bounding and reduction technique. MINKNAP algorithm requires all the coefficients of the knapsack problem being integer. However, in our case the objective function coefficients of the problems in the row generation routine can be fractional. Ceselli and Righini [16] modified the MINKNAP algorithm to deal with real coefficients allowing us to solve the problem with the given accuracy.

On Step 2 Kaparis and Letchford [26] have suggested the two following ways to enhance the row generation:

(1) They try to solve the knapsack problems heuristically, and only call the exact routine when the heuristic fails. They run the simple greedy heuristic with improvements by a local search.
(2) They also try to "warm-start" the separation LP (9) by adding to the set

of initial constraints the solutions which were generated in the previous call of the separation routine.

We have tried both of these enhancements, but they do not affect the computational time in our implementation.

2.3. *Numerical errors*

The row generation method for solving LP-problems (Step 1) and the knapsack problem with fractional coefficients of the objective vector (Step 2) may include rounding errors. When searching for cuts, the effect of these errors may be significant, thus leading to inequalities to be weak or even invalid. To generate safe cutting planes, the obtained inequalities are post-processed to get the equivalent cuts with integer coefficients and verified validity.

So, first of all we have to implement a scaled procedure, which finds a minimum positive multiplier making all the coefficient of $\bar{\pi}$ integer. It has been done by solving a special integer programming problem in [3, 9]. Before solving the integer problem, Kaparis and Letchford [26] have suggested the following heuristic rule, which works in over 90% of cases. Divide $\bar{\pi}$ by $\min\limits_{i=\overline{1,k}}\{\bar{\pi}_i - \lfloor\bar{\pi}_i\rfloor : \bar{\pi}_i > 0\}$, and check if the resulting vector is integral (within a tolerance of 10^{-5}).

Our empirical study has shown, that the simplest enumeration of multipliers from 2 to 10^4 and checking the integrality within the tolerance of 10^{-5} makes this procedure very fast, as it will be illustrated in computational results in Section 3. If the integral vector is not found in this range of multipliers, we skip this inequality from the consideration to avoid further problems with the arithmetic overflow.

Let $\hat{\pi}$ be a resulting integer vector. To prevent rounding errors and be sure that the inequality is valid, the right hand side is computed as

$$\hat{\pi}_0 = \max\{\hat{\pi}^T x : x \in X(a, b, \bar{x})\}. \tag{10}$$

Previously in [3, 9] this problem was solved by MINKNAP algorithm. We suggest solving it by the dynamic programming algorithm, which will be discussed in the next subsection, since it is closely linked with the our lifting procedure. All the coefficients in the knapsack problem are integer, so the solution is exact and we are sure that the inequality $\hat{\pi}^T x \le \hat{\pi}_0$ is valid for $P(a, b, \bar{x})$ and it is passed to the sequential lifting procedure if it remains violated by $\bar{x}$.

2.4. *Sequential lifting*

As it has been illustrated in [3], the most time consuming part on the separation routine is the sequential lifting, therefore we pay more attention to this issue.

Initially we have the solution of problem (10) and due to the lifting theorem we have to do the following:

Up-lifting. If we have that $\bar{x}_{k+1} = 0$ then the lifted inequality is

$$\sum_{j=1}^{k} \hat{\pi}_j x_j + (\hat{\pi}_0 - \pi^0_{k+1}) x_{k+1} \le \hat{\pi}_0,$$

where

$$\pi^0_{k+1} = \max\left\{\sum_{j=1}^{k}\hat{\pi}_j x_j : \sum_{j=1}^{k} a_j x_j \le \bar{b} - a_{k+1}, x \in \{0,1\}^k\right\}. \qquad (11)$$

Down-lifting. If we have that $\bar{x}_{k+1} = 1$ then the lifted inequality is

$$\sum_{j=1}^{k}\hat{\pi}_j x_j + (\pi^1_{k+1} - \hat{\pi}_0)x_{k+1} \le \pi_{k+1},$$

where

$$\pi^1_{k+1} = \max\left\{\sum_{j=1}^{k}\hat{\pi}_j x_j : \sum_{j=1}^{k} a_j x_j \le \bar{b} + a_{k+1}, x \in \{0,1\}^k\right\}. \qquad (12)$$

Let us look more attentively at problems (10)-(12). First of all we have to mention that we are not interested in their solution vectors, we only need the optimal values. Moreover we have the knapsack problems which have only different capacities. So if we solve the knapsack problem (10) with maximum possible capacity $\bar{b}$ by the dynamic programming algorithm, we immediately have the optimal values of all the problems (10)-(12).

Actually, we can use the dynamic programming algorithm in the following way (see for example [27, 30]). Let $z_i(\beta)$ be the optimal values of knapsack problem (10) with i variables and capacity β for $i = \overline{1,k}$ and $\beta = \overline{0,\bar{b}}$. It can be computed by the following recursion formulas:

$$z_1(\beta) = \begin{cases} 0, & \text{if } \beta < a_1 \\ \hat{\pi}_1, & \text{if } \beta \ge a_1 \end{cases}$$

$$z_i(\beta) = \begin{cases} z_{i-1}(\beta), & \text{if } \beta < a_1 \\ \max\{z_{i-1}(\beta), z_{i-1}(\beta - a_i) + \hat{\pi}_i\}, & \text{if } \beta \ge a_1 \end{cases} \qquad (13)$$

We immediately have that

$$\hat{\pi}_0 = z_k(\bar{b}), \quad \pi^0_{k+1} = z_k(\bar{b} - a_{k+1}), \quad \pi^1_{k+1} = z_k(\bar{b} + a_{k+1}).$$

To lift the next variable we have to assign

$\hat{\pi}_{k+1} = \hat{\pi}_0 - \pi^0_k$ in the case of up-lifting,
$\hat{\pi}_{k+1} = \pi^1_{k+1} - \hat{\pi}_0$, $\hat{\pi}_0 = \pi^1_{k+1}$, $\bar{b} := \bar{b} + a_{k+1}$ in the case of down-lifting,

and compute $z_{k+1}(\beta)$ for $\beta = \overline{0,\bar{b}}$ by (13).

Thus, the complexity of solving problem (10) and lifting all the nonfractional variables is $O(n \cdot \bar{b})$ and it requires only $O(\bar{b})$ of memory because we do not need solution vectors. This procedure allows us to drastically reduce the computational time in comparison to solving problems (10)-(12) separately by the MINKNAP algorithm as in [3, 9, 26].

The lifting procedure is sequential and the resulting valid inequality depends on the order of lifting. We have tried different orders and their comparison is given in Section 3.

3. Computational results

The proposed algorithm has been implemented in C/C++ and linked with IBM ILOG CPLEX optimizer 12.1[1] library for solving linear programming problems and as a branch-and-cut framework. Computational experiments were carried out on an Intel Core 2Quad CPU $2.6GHz$ workstation with $4Gb$ of RAM, under Windows $XP64$. We did not use multithreads, so computations are limited to a single core.

In Subsection 3.1 different variants of our approach are tested on the Generalized Assignment Problem (GAP) in order to find the best one. There is also comparison with CPLEX solver and the state-of-the-art approach proposed by Posta et al. [35]. In the next subsection, the computational experiments on the Multilevel Generalized Assignment Problem, Capacited p-Median Problem and Capacitated Network Location Problem are presented.

Kaparis and Letchford [26] have tested the exact and different other knapsack inequalities for non specially structured instances: instances taken from MIPLIB[1] and multi-dimensional knapsack problem taken from ORLIB [7]. We have tried them, but do not give any computational results because the exact knapsack separation has shown to be useless from a practical point of view. Actually, either the separation takes much more time than finding an optimal solution by CPLEX or the closed gap is negligible. Unfortunately, their code is not still available we have not been able to do any comparison.

Our code, problems instances are publicly available and can be found at iv.icc.ru/papers.html.

3.1. *Generalized Assignment Problem*

Let $M = \{1, \dots, m\}$ be a set of machines and let $N = \{1, \dots, n\}$ be a set of tasks to be assigned to the machines from M. Let c_{ij} be the cost of assigning the task j to the machine i. Let d_{ij} be the amount of resource required by the machine i to perform the task j. Each machine has a limited amount of resources available. Let u_i be the capacity of the machine $i \in M$.

The *Generalized Assignment Problem* (GAP) is to find a minimum assignment cost of the tasks N to the machines M satisfying the constraint that the total amount of resources required by each machine $i \in M$ does not exceed its capacity u_i.

Let x_{ij} be a binary variable expressing the assignment of the task $j \in N$ to the machine $i \in M$. The GAP can be formulated as:

$$\min_x \sum_{i \in M} \sum_{j \in N} c_{ij} x_{ij}$$

$$\sum_{i \in M} x_{ij} = 1, \quad j \in N \tag{14}$$

$$\sum_{j \in N} d_{ij} x_{ij} \leq u_i, \quad i \in M \tag{15}$$

$$x_{ij} \in \{0, 1\}, \quad i \in M, j \in N \tag{16}$$

Constraints (14) require that each task must be assigned to a machine. Capacity constraints (15) enforce the condition that the amount of resources required by the

[1]http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/
[1]http://miplib.zib.de/

	ncuts		SEP(sec)		RG(%)		SCALE(%)		LIFT(%)	
n	old	new	old	new	old	new	old	new	old	new
100	3172	3172	16.8	12.0	69.9	99.0	16.3	0.0	13.8	1.0
200	3977	3977	36.2	19.8	54.1	97.9	9.7	0.0	36.2	2.1
400	8257	8242	112.3	29.1	24.0	95.8	6.4	0.1	69.6	4.1
900	12253	12316	626.4	52.3	7.7	89.5	2.0	0.0	90.4	10.3
1600	17117	17172	2196.2	68.2	2.5	81.0	1.0	0.0	96.5	18.9

Table 1. Comparison with previous version

tasks assigned to the machine i does not exceed its capacity u_i and we can apply our separation routine over them.

Because of its computational difficulty, GAP is a challenging integer programming problem, which is widely addressed in literature. A short survey and comprehensive list of references concerning this problem can be found in recent papers [3, 35].

We consider test instances included in the OR-Library [8]. The test-bed consists of three types of instances (C, D, E) with size from 5×100 to 80×1600. They are named according to their type and size, e.g. d05100 is an instance of type D with $m = 5$ machines and $n = 100$ tasks.

The computational results of the previous version of exact knapsack separation have been given for GAP and CPMP in [3, 9]. The branch-and-cut algorithm with this cutting plane procedure outperforms CPLEX and the branch-and-price algorithms [16, 33], which were the state-of-the-art approaches at that moment.

The results of comparison the previous version of cutting plane algorithm and version considered in this paper is presented in Table 1. The main differences between them are in the scaling procedure described in Section 2.3 and lifting from Section 2.4. We give the total results of instances with the same knapsack size, i.e. when the number of jobs is the same. We limit the number of generated cuts if their number exceeds 3000 on some iteration of cutting plane procedure then we stop. There are nine instances of each dimension. The notations in the table are the following:

n — knapsack size;
ncuts — total number of cuts generated for all instances of corresponding dimension;
SEP(sec) — total time spent in separation routine in seconds;
RG(%) — time spent in the row generation relative to the computational time of complete separation in routine in percents;
SCALE(%) — time spent in the scaling procedure in percent;
LIFT(%) — time spent in the sequential lifting in percent;
old — results of previous version from [3];
new — results of the version presented in this paper.

As you can see in the table and as it has been pointed out in [3], the bottleneck of the exact separation routine is the lifting procedure. If we do not focus on small instances, we can conclude that we noticeably reduce the computational time. It has been done mainly thanks to the enhanced lifting procedure, which evidently shows its efficiency. The computational time of the scaling procedure has become negligible, which improves the results as well.

Thus the row generation is the most time consuming part of the separation routine. It is known that MINKNAP algorithm is very efficient for different knapsack instances [27, 34]. Therefore we have decided to check it for our case. Actually, it has been mentioned in [3] that the size of knapsack problems is usually small in row generation, and it might happen that the simple algorithms from textbooks [27, 30] are more effective than the sophisticated MINKNAP. The computational results are given in Table 2, where the ratio of the computational time spent in knapsack

	KNAP/RG(%)			
n	MINKNAP	DP	DPR	BB
100	4.2	12.3	5.4	6.8
200	4.3	14.0	8.9	6.7
400	3.5	13.8	8.6	6.2
900	3.4	10.9	8.5	8.3
1600	3.4	10.2	6.3	14.3

Table 2. Comparison of different knapsack algorithms

	ncuts		SEP time		CP time	
n	EKS	LCI+EKS	EKS	LCI+EKS	EKS	LCI+EKS
100	3172	3103	11.988	11.455	14.623	14.125
200	3977	3920	19.812	20.958	25.547	26.968
400	8242	8437	29.091	29.644	76.735	93.343
900	12316	12574	52.342	53.381	165.269	167.189
1600	17172	17464	68.179	64.287	606.124	628.874

Table 3. Comparison of results with the LCI separation

algorithm and total time of row generation (in percents) is given for the following knapsack algorithms: **MINKNAP** is the modification of Pisinger's MINKNAP algorithm, **DP** is a genetic dynamic programming algorithm, **DBR** is the dynamic programming algorithm with eliminating of dominated states, **BB** is the simplest branch-and-bound algorithm based on the LP relaxation bounds. As the reader can see in the table, the MINKNAP remains the most efficient for our case as well.

We can also mention from Table 2 that the knapsack algorithm does not take much time in the row generation, so the main part of time is spent in optimising the separation LP by CPLEX, which we cannot improve directly. To draw this obstacle Kaparis and Letchford [26] have recommended to separate the Lifted Cover Inequalities (LCI) before the exact separation routine, since the LCI separation is very fast and effective. We have also tried this strategy. The results are presented in Table 3, where in columns **EKS** the results are given for cutting plane algorithms only with the exact knapsack separation, **LCI+EKS** — results with the separation of LCI. **SEP time** and **CP time** denoting the computational time of the separation routines and the complete cutting plane algorithm correspondingly are given in seconds. It is readily seen that the separation time is similar, but the complete cutting plane takes more time with LCI since more cuts are generated.

To complete the tuning of the exact separation routine we have tested different orders of variables in sequential lifting. The following four strategies have been tested:

LIFT1 — the variables, which are fixed to one, are down-lifted first, and then the variables, which are fixed to zero, are up-lifted. This strategy has been used in [3, 9] and in the previous experiments of this paper. This lifting order keep the facet-inducing property, i.e. if the initial inequality is facet-inducing then the resulting one is facet-inducing as well.
LIFT2 — Kaparis and Letchford [26] have suggested to firstly consider the variables for up-lifting for which $a_i > \beta$. It immediately follows that their lifting coefficients can be set to 0 and it can make the lifting procedure faster.
LIFT3 — Variables are lifted in increasing order of their reduced costs.
LIFT4 — Variables are lifted in decreasing order of their reduced costs, which is a counterpart of the previous strategy.

The number of generated cuts, computational time in separation and cutting plane algorithms for all the considered strategies are presented in Table 4. In contrast to [26], **LIFT2** has not given the best results. In our opinion it has happened because the lifting procedure is quite fast in our version and this strategy does give any advantage. On the other hand, **LIFT1** ensures us to have the facet-inducing inequality and generates the less number of cuts, so this lifting order is used in the

	ncuts				SEP time (secs)			
n	LIFT1	LIFT2	LIFT3	LIFT4	LIFT1	LIFT2	LIFT3	LIFT4
100	3172	3685	2982	3786	12.0	13.5	10.9	14.9
200	3977	4611	3741	4621	19.8	24.7	20.4	25.1
400	8242	9640	7886	9555	29.1	42.4	29.4	41.5
900	12316	14812	11784	14834	52.3	81.4	53.0	83.7
1600	17172	19445	16018	19640	68.2	117.1	69.9	110.4

	CP time (secs)			
n	LIFT1	LIFT2	LIFT3	LIFT4
100	14.6	16.7	13.1	18.3
200	25.5	31.7	25.9	31.9
400	76.7	162.1	90.7	151.4
900	165.3	264.5	171.5	277.2
1600	606.1	845.2	605.9	1710.8

Table 4. Comparison of lifting orders

further experiments as well.

Now let us consider how the generated cuts can be useful for finding optimal solutions. The results of the previous version of our approach are presented in [3] and they are the best in comparison with CPLEX and the state-of-the-art branch-and-price algorithms [33]. We are updating these results with the results of our new version.

The computational results of the cut-and-branch (i.e. the separation routing is used only in the root node of branch-and-bound tree) in columns **C&B** in comparison with CPLEX (**CPX**) and an approach **POSTA** proposed by Posta et al. [35], which consists of solving a series of decision problems by a lagrangean branch-and-bound algorithm with a few variable fixing rules, are presented in Table 5. **OPT** is an optimal value and the computational time is given for two cases: **Proof** — when the methods are run with the given optimal values, i.e. we just prove that these values are optimal; **Search** — when the methods are run from the scratch. The results are given on instances, for which it is possible to find an optimal solution with the time limit of two hours and given memory. The blanc cells of the table mean that the corresponding method cannot solve the problem with the given limits. We can see that the proposed cutting planes sufficiently improve CPLEX branch and bound, but they are inferior to the approach specially developed for GAP except of two cases *c60900, c201600*. Using the branch-and-cut (i.e. the separation routine is used in any node of branch-and-bound tree) we can proof the optimality of the solution for *c401600* in 1845.1 seconds while POSTA do it in 7835.27 seconds.

3.2. *Multilevel Generalized Assignment Problem*

In the context of large manufacturing systems, a variant of generalized assignment problem was suggested in [24]. In such systems machines can perform tasks in different "levels", which entails different costs and different amount of resources required. In addition to the GAP, a set $K = \{1, \ldots, k\}$ of levels is considered. Consequently c_{ijk} is the cost of assigning the task j to the machine i at the level k, d_{ijk} is the corresponding amount of resources required and a binary variable x_{ijk} expresses such kind of assignment. The Multilevel Generalized Assignment

		Proof			Search	
Name	OPT	C&B	POSTA	CPX	C&B	POSTA
c05100	1931	0.2	0.0	0.1	0.3	0.0
c10100	1402	0.3	0.1	0.5	0.7	0.1
c20100	1243	0.2	0.1	0.4	0.4	0.2
d05100	6353	5.0	0.8	37.2	19.9	3.3
d10100	6347	203.4	8.6		1377.2	26.7
e05100	12681	0.9	0.2	1.2	1.0	0.7
e10100	11577	2.5	0.6	17.4	4.1	1.9
e20100	8436	3.3	0.8	0.0	15.7	1.6
c05200	3456	1.0	0.2	0.2	0.9	0.3
c10200	2806	1.1	1.5	5.0	3.1	2.0
c20200	2391	0.4	0.1	4.0	2.8	0.5
d05200	12742	6.5	0.8	307.5	572.5	1.4
e05200	24930	0.6	0.1	0.1	1.0	0.4
e10200	23307	5.5	1.9	20.3	4.6	3.2
e20200	22379	2.2	0.4	120.9	3.6	0.6
c10400	5597	3.3	1.7	14.3	3.3	2.9
c20400	4782	10.5	6.3	3871.7	16.7	16.3
c40400	4244	0.8	1.9	21.2	3.2	6.4
e10400	45746	1.5	0.4	1.5	14.2	0.6
e20400	44877	3.0	0.6	2946.8	21.2	1.3
e40400	44561	138.2	36.5		203.3	67.5
c15900	11340	4.0	1.7	300.6	411.4	11.1
c30900	9982	2.8	2.2		393.6	240.8
c60900	9326	41.1	131.6			
e15900	102421	2.0	0.6	564.4	12.0	2.3
e30900	100427	20.1	0.5		58.9	2.9
e60900	100149	104.5	877.6			
c201600	18802	2.8	2.3		1867.2	3744.9
e201600	180645	17.7	23.1		167.1	25.8
e401600	178293	10.8	2.4		578.5	156.8
e801600	176820	66.6	7.8		1076.4	48.7

Table 5. Results on GAP instances

Problem (MGAP) can be formulated as:

$$\min_x \sum_{i\in M}\sum_{j\in N}\sum_{k\in K} c_{ijk}x_{ijk}$$

$$\sum_{i\in M}\sum_{k\in K} x_{ijk} = 1, \quad j \in N \tag{17}$$

$$\sum_{j\in N}\sum_{k\in K} d_{ijk}x_{ijk} \le u_i, \quad i \in M \tag{18}$$

$$x_{ijk} \in \{0,1\}, \quad i \in M, j \in N, k \in K \tag{19}$$

Heuristic approaches for the MGAP was proposed in [22, 28]. A branch and cut algorithm was able to solve only small instances [31]. The most recent results were obtained in [18] by the branch and price (B&P) algorithm on instances up to 80 machines, 400 task and 5 levels. Nowadays it is a state-of-the-art exact approach for the MGAP.

The B&P algorithm was tested on generated instances of three types (C, D, E). The instances of type C are very easy and they can be solved by CPLEX in a few second. The instances of type D are very hard to solve, and they are not tractable by our approach. Therefore we tested our approach on instances of type E. The results are presented in Table 6. We skip the "easy" instances, i.e. the instances for those CPLEX can find the optimal solutions in less than 10 seconds. Only the non-easy instances will be considered in the further experiments. The computational time of B&P is given which was obtained on a PC with Intel P4 1.6GHz CPU and 512 MB of RAM.

By the same way as in [18] we tested our approach on large-scale instances which were obtained from the GAP instances with 400, 900 and 1600 tasks. These tasks

					Proof		Search	
Name	m	n	k	OPT	C&B	CPX	C&B	CPX
E1010003-5	10	100	3	10757	1.7		2.4	
E1010005-3	10	100	5	10576	1.1	85.9	4.4	10.7
E1010005-4	10	100	5	10355	1.0	26.5	2.0	
E2010003-1	20	100	3	7136	1.2		2.9	
E2010003-2	20	100	3	7352	4.1		11.8	
E2010003-3	20	100	3	7239	5.6		15.2	
E2010003-4	20	100	3	7289	1.9		4.5	1666.5
E2010003-5	20	100	3	7494	1.5		2.7	523.1
E2010004-1	20	100	4	6373	5.5		8.5	
E2010004-2	20	100	4	6374	3.3		10.9	966.6
E2010004-3	20	100	4	7050	3.7		22.1	
E2010004-4	20	100	4	7090	2.4		4.7	509.1
E2010004-5	20	100	4	7090	2.5	573.5	3.7	136.5
E2010005-1	20	100	5	6074	3.9		12.3	
E2010005-2	20	100	5	6213	2.8		6.0	
E2010005-3	20	100	5	6554	5.3		7.1	
E2010005-4	20	100	5	6338	3.3		11.2	
E2010005-5	20	100	5	5975	5.6		22.6	
E3010003-1	30	100	3	4224	12.5		70.8	
E3010003-2	30	100	3	4647	48.6		47.7	
E3010003-3	30	100	3	4528	27.7		62.1	
E3010003-4	30	100	3	4353	21.7		81.9	
E3010003-5	30	100	3	4298	28.0		102.0	
E3010004-1	30	100	4	4258	8.2		23.2	
E3010004-2	30	100	4	4286	22.7		85.2	
E3010004-3	30	100	4	4449	11.5		30.3	
E3010004-4	30	100	4	4213	25.1		78.2	
E3010004-5	30	100	4	4215	13.7		57.2	
E3010005-1	30	100	5	3888	21.6		84.5	
E3010005-2	30	100	5	4015	10.3		55.0	
E3010005-3	30	100	5	4152	11.8		30.6	
E3010005-4	30	100	5	3988	9.4		24.5	
E3010005-5	30	100	5	4090	6.0		18.3	
E1520004-1	15	200	4	25320	2.7		5.7	1113.4
E1520004-2	15	200	4	27622	1.2	1679.0	3.1	49.0
E1520004-3	15	200	4	26368	1.7	364.5	4.9	20.5
E1520004-4	15	200	4	23177	2.6		5.2	1502.5
E1520004-5	15	200	4	23383	2.2		6.3	90.2
E1520005-1	15	200	5	22117	4.6		10.8	282.6
E1520005-2	15	200	5	21160	2.3		4.9	277.4
E1520005-3	15	200	5	21393	3.1		9.3	25.2
E1520005-4	15	200	5	21138	2.7	916.2	7.4	47.5
E1520005-5	15	200	5	21055	1.9	748.0	5.8	95.5
E3020004-1	30	200	4	17171	7.4		16.4	
E3020004-2	30	200	4	17258	6.2		17.9	
E3020004-3	30	200	4	16658	25.7		57.4	
E3020004-4	30	200	4	16851	8.0		42.9	
E3020004-5	30	200	4	18196	4.5		9.6	
E3020005-1	30	200	5	16635	6.7		18.8	
E3020005-2	30	200	5	17461	8.1		27.2	
E3020005-3	30	200	5	17233	10.8		24.3	
E3020005-4	30	200	5	16455	5.7		16.0	
E3020005-5	30	200	5	16202	10.1		25.0	

Table 6. Results on MGAP instances

are divided in 2, 3 and 4 levels with the corresponding adjustments of the capacities. In Table 8 the results on instances of type C and E are presented and they are also compared with the B&P.

From the presented results it is evidently seen that the proposed cutting planes substantially increase the efficiency of the CPLEX solver and, in spite of the fact that our PC can be in 3-4 times faster, make it very competitive with the B&P algorithm.

3.3. *Capacitated p-Median Problem*

Let $G(V, A)$ be a complete digraph, with node set $V = \{1, \ldots, n\}$, arc set $A = \{ij : \ i \in V, \ j \in V\}$ and costs c_{ij}, $ij \in A$, usually called "distances". A demand d_i

m	n	k	C&B	B&P
10	100	3	3.1	63.5
10	100	4	1.2	10.7
10	100	5	2.3	14.4
20	100	3	7.4	53.9
20	100	4	10.0	80.7
20	100	5	11.8	68.6
30	100	3	72.9	598.0
30	100	4	54.8	274.4
30	100	5	42.6	190.9
15	200	4	5.0	975.2
15	200	5	7.6	1037.2
30	200	4	28.8	199.2
30	200	5	22.2	271.3

Table 7. Comparison with the branch and price on MGAP instances

					Proof		Search		
Name	m	n	k	OPT	C&B	CPX	C&B	CPX	B&P
c10400	10	200	2	2378	0.5	0.6	2.0	28.9	757.7
e10400	10	200	2	22172	1.8	63.7	7.6	13.2	17473.8
e20400	20	200	2	22080	2.9		4.8	327.7	121.9
e40400	40	200	2	22031	3.1		10.1		151.9
e15900	15	300	3	33334	3.6	3111.7	13.9	58.2	6607.2
e30900	30	300	3	33097	12.3		50.7		1759.1
e60900	60	300	3	33106	4.2		10.8	522.5	313.6
e201600	20	400	4	44160	9.0		71.2	993.3	12996.3
e401600	40	400	4	43910	22.3		64.5		3188.5
e801600	80	400	4	43762	4.9	8.6	8.3	28.7	2685.8

Table 8. Results on large-size MGAP instances

and a capacity q_i are associated with each node $i \in V$. The Capacitated p-Median problem (CPMP) consists of finding p nodes (*the median nodes*) minimizing the total distance to the other nodes of graph, with the additional requirement that the total demand of the nodes assigned to each median node i does not exceed its capacity q_i.

Let y_i be the binary variable associated with the node $i \in V$ ($y_i = 1$ if node i is a median, 0 otherwise) and let x_{ij} be the binary variable associated with the arc $ij \in A$ ($x_{ij} = 1$ if node j is assigned to median node i, 0 otherwise). The Integer Linear Programming formulation of CPMP is:

$$v = \min \sum_{ij \in A} c_{ij} x_{ij} \tag{20}$$

s.t.

$$\sum_{i \in V} x_{ij} = 1, \quad j \in V \tag{21}$$

$$\sum_{i \in V} y_i = p \tag{22}$$

$$\sum_{j \in V} d_j x_{ij} \leq q_i y_i, \quad i \in V \tag{23}$$

$$y_i \in \{0,1\}, \quad i \in V \tag{24}$$

$$x_{ij} \in \{0,1\}, \quad ij \in A \tag{25}$$

Equality constraints (21) impose that each node is served by one node only. The number of median nodes is enforced by the equality (22). Capacity constraints (23) ensure that the total demand of the nodes assigned to the median node i cannot exceed its capacity q_i. Variable Upper Bound constraints $x_{ij} \leq y_i, ij \in A$ could be added to strengthen the formulation (20)-(25). Formulation (20)-(25) is the same used in [16] and it is more general than that used in [6], where it is supposed that

			Proof		Search	
Name	n	OPT	B&C	CPX	B&C	CPX
dd1	100	17289.0	2.6	15.6	3.5	42.0
dd2	200	33270.9	23.2	8.9	37.8	33.3
dd3a	300	45335.2	33.3	43.8	50.5	73.4
dd3b	300	40635.9	13.5	10.8	5.6	21.0
dd4a	402	61925.5	262.3	173.0	379.9	423.8
dd4b	402	52458.0	141.5	37.0	147.4	62.4

Table 9. Results on CPMP instances from [29]

$y_i = x_{ii}, i \in V$. So it is assumed that the graph contains self loops $ii \in A$ for all $i \in V$ and that a median node could be assigned to another one.

The previous version of our approach has been tested and compared with the CPLEX and a state-of-the-art branch-and-price algorithm [16] on instances presented ibidem (instances with names cpmpXX) and in [29] (instances with names ddXX). As it has outperformed the branch-and-price (and actually, CPLEX has outperformed the branch-and-price as well) and the new version cannot be less efficient, we compare our result only with the current version of CPLEX. The results are given in Tables 9-11 with the same notation as for the GAP. In most cases the cut-and-branch algorithm is more effective than CPLEX.

Some blank cells can be filled using the branch-and-cut algorithm, whose results are given in table 12. We have only cpmp32 and cpmp34 of type β unsolved to optimality yet.

3.4. *Capacitated Network Location Problem*

By generalizing the CPMP, Ceselli et al. [17] generated different problems, which they named as the Capacitated Network Location Problems (CNLP). Their first test bed was based on the single source capacitated facility location problem instances of Holmberg et at.[25], Diaz and Fernández [20]. By modifying the fixed costs and adding the cardinality constraints (like in the CPMP) 4 types of problem instances were generated (a, b, c, d). Holmberg instances are easy and they are omitted. In Table 13 there are only the Diaz instances for which CPLEX cannot find the optimal solutions in 10 seconds. Also the results of the B&P algorithm are given, which are taken from the paper [17] and which were obtained on the same PC as for the MGAP.

There are also instances based on the CPMP in [17], but they are not available and, thus, are not included into the consideration.

4. Conclusions

In this paper we are reporting on the computational experiments on a new implementation of the exact separation over the 0-1 knapsack polytope. The careful elaboration of all algorithm elements (especially the lifting procedure) and their correct tuning make this approach much more efficient. It was proved on the following problem whose constraints have a distinct knapsack structure: the Generalized Assignment Problem (GAP), Multilevel Generalized Assignment Problem (MGAP), Capacitated p-Median Problem, Capacitated Network Location Problem.

The computational results apparently show that for these problem the proposed approach can be a very competitive counterpart to the branch and price approach. Another advantage of our approach that it deals with the explicit standard formulation of these problems, containing only the original variables. The user can easily use our separation procedure in the standard Branch and Cut frameworks, like IBM ILOG CPLEX, without deep diving in the exigeant implementation of

				Proof		Search	
Name	n	type	OPT	C&B	CPX	C&B	CPX
cpmp08	50	α	820	23.2	11.4	20.8	19.6
cpmp14	100	α	982	15.4	5.0	30.8	19.4
cpmp15	100	α	1091	13.0	10.6	21.1	14.5
cpmp18	100	α	1043	13.6	4.0	14.8	19.0
cpmp20	100	α	1005	211.4	204.5	648.1	428.7
cpmp21	150	α	1288	18.6	8.6	39.3	49.7
cpmp22	150	α	1256	41.2	43.3	107.9	220.7
cpmp23	150	α	1279	2.8	7.1	6.5	11.9
cpmp24	150	α	1220	6.8	14.6	11.7	16.5
cpmp26	150	α	1264	59.9	11.6	76.4	28.7
cpmp27	150	α	1323	181.8	172.1	234.4	179.2
cpmp31	200	α	1378	27.3	9.8	42.9	24.8
cpmp32	200	α	1424	360.3	1324.0	531.0	3721.8
cpmp33	200	α	1367	35.3	16.9	126.5	65.8
cpmp34	200	α	1385	163.0	57.3	241.1	609.5
cpmp35	200	α	1437	78.3	33.7	206.2	114.7
cpmp36	200	α	1382	13.1	10.5	36.0	69.1
cpmp37	200	α	1458	28.0	8.4	51.0	22.7
cpmp38	200	α	1382	127.0	59.1	338.3	353.5
cpmp39	200	α	1374	20.9	6.8	87.7	13.7
cpmp40	200	α	1416	13.2	11.2	86.1	24.3
cpmp04	50	β	384	48.0	25.0	46.0	30.5
cpmp07	50	β	445	51.5	29.1	18.4	22.3
cpmp09	50	β	436	4.0	6.6	8.2	10.6
cpmp10	50	β	461	16.7	21.3	33.0	37.3
cpmp11	100	β	544	20.4	69.0	39.3	83.1
cpmp12	100	β	504	4.9	6.4	16.3	17.7
cpmp13	100	β	555	40.3	74.5	131.1	90.8
cpmp14	100	β	544	21.3	21.1	28.8	57.0
cpmp15	100	β	583	12.6	46.3	27.2	79.7
cpmp16	100	β	534	261.7	1019.7	340.4	594.5
cpmp18	100	β	508	4.3	6.6	11.7	10.5
cpmp19	100	β	551	116.5	203.0	164.6	212.2
cpmp20	100	β	*555*				
cpmp21	150	β	681	14.1	20.3	18.0	20.3
cpmp22	150	β	660	75.2	1562.3	218.0	1272.6
cpmp23	150	β	663	611.8	5357.6	811.4	4132.7
cpmp24	150	β	594	21.3	71.8	22.6	50.0
cpmp26	150	β	653	12.4	20.6	21.8	15.1
cpmp27	150	β	736	1831.0		604.6	5969.5
cpmp28	150	β	644	37.6	30.1	47.4	26.4
cpmp29	150	β	649	11.4	14.2	24.3	10.3
cpmp30	150	β	630	24.7	63.7	46.9	132.6
cpmp31	200	β	727	258.9	2600.6	225.3	4317.2
cpmp32	200	β	*832*				
cpmp33	200	β	713	636.7		791.3	
cpmp34	200	β	*816*				
cpmp35	200	β	746	1308.4		776.2	6252.1
cpmp36	200	β	701	45.4	472.5	105.1	151.6
cpmp37	200	β	753	14.4	35.9	35.3	79.6
cpmp38	200	β	746	943.7	5020.3	762.0	
cpmp39	200	β	722	53.5	60.9	111.1	716.6
cpmp40	200	β	*763*				

Table 10. Results on CPMP instances, types α and β

the Branch and Price algorithm.

Of course, the proposed approach cannot be the best for all cases even for the considered problems. For example, it loses the specially designed algorithm for the GAP, and it is not efficient for the GAP and MGAP instances of type D. But it can be useful tool to couple with others methods, which take more advantage from the study of specific problem structure. Such kind of the study can open new directions for the further investigations.

Acknowledgements

The authors wish to thank Alberto Ceselli, Marcus Posta for their codes and test instances and Pasquale Avella for helpful comments. The research of I. Vasilyev is

				Proof		Search	
Name	n	type	OPT	C&B	CPX	C&B	CPX
cpmp08	50	γ	353	18.3	48.1	35.5	101.9
cpmp10	50	γ	390	9.8	42.9	12.5	53.1
cpmp12	100	γ	391	17.1	71.0	27.2	61.0
cpmp14	100	γ	447	11.8	47.0	14.8	46.5
cpmp16	100	γ	447	34.2	214.0	140.5	109.7
cpmp18	100	γ	456	12.3	110.3	22.0	87.8
cpmp19	100	γ	445	19.4	32.3	20.4	26.3
cpmp20	100	γ	460	12.4	87.8	34.6	252.5
cpmp21	150	γ	599	152.4	995.0	206.3	733.1
cpmp22	150	γ	561	201.5	3514.7	570.1	3988.2
cpmp23	150	γ	564	32.6	597.0	90.0	800.0
cpmp24	150	γ	505	21.0	113.8	98.9	108.0
cpmp26	150	γ	540	3.6	166.3	7.9	26.4
cpmp27	150	γ	579	40.6	234.6	104.4	199.6
cpmp28	150	γ	503	5.2	13.9	5.8	13.3
cpmp29	150	γ	545	47.6	1006.1	60.9	473.0
cpmp30	150	γ	502	21.3	467.1	51.0	124.5
cpmp31	200	γ	575	5.5	63.4	6.8	21.8
cpmp32	200	γ	*724*				
cpmp33	200	γ	618	1615.0		3511.7	
cpmp34	200	γ	709			3288.8	
cpmp35	200	γ	611	349.5	3724.3	525.0	5209.9
cpmp36	200	γ	580	45.4	390.7	110.1	692.8
cpmp37	200	γ	631	1625.4	5517.7	3845.8	
cpmp38	200	γ	608	94.6	1320.9	122.3	350.3
cpmp39	200	γ	589	216.2	5720.9	713.5	724.1
cpmp40	200	γ	630	183.9		498.8	
cpmp08	50	δ	312	3.8	21.1	7.6	16.1
cpmp09	50	δ	412	3.4	38.5	6.8	41.1
cpmp10	50	δ	458	8.9	118.0	8.9	254.8
cpmp11	100	δ	415	5.8	58.2	17.7	21.9
cpmp12	100	δ	377	15.4	233.0	29.1	82.3
cpmp14	100	δ	421	9.0	29.7	22.5	20.5
cpmp15	100	δ	496	7.4	59.7	12.0	10.6
cpmp16	100	δ	428	2.4	85.1	5.7	24.1
cpmp17	100	δ	440	13.1	100.6	40.7	31.8
cpmp18	100	δ	450	16.1	66.4	59.1	31.4
cpmp19	100	δ	450	8.7	92.0	20.2	62.9
cpmp20	100	δ	486	19.1	173.4	65.9	330.2
cpmp21	150	δ	552	15.3	116.5	28.4	38.5
cpmp22	150	δ	601	166.5		250.5	
cpmp23	150	δ	555	39.4	1298.6	121.5	988.6
cpmp24	150	δ	487	238.1	2590.2	200.2	269.8
cpmp25	150	δ	436	10.3	276.1	20.8	44.2
cpmp26	150	δ	512	5.6	36.4	7.3	26.1
cpmp27	150	δ	*743*				
cpmp28	150	δ	471	21.2	413.7	71.1	55.1
cpmp30	150	δ	444	5.6	109.7	7.0	19.8
cpmp31	200	δ	541	258.2		2505.7	1844.0
cpmp32	200	δ	*802*				
cpmp33	200	δ	557	90.1	1578.2	149.2	344.8
cpmp34	200	δ	845	1114.7		1586.7	
cpmp35	200	δ	552	17.3	906.6	32.5	5245.7
cpmp36	200	δ	551	75.3	3105.3	154.8	1774.8
cpmp37	200	δ	594	360.1	6795.2	381.6	1624.0
cpmp38	200	δ	592	63.3	2976.3	43.0	1820.0
cpmp39	200	δ	540	166.1	1803.5	238.0	595.1
cpmp40	200	δ	588	1316.4		1349.0	

Table 11. Results on CPMP instances, types γ and δ

Name	n	type	OPT	Proof B&C
cpmp20	100	β	555	14248.2
cpmp40	200	β	762	38480.1
cpmp32	200	γ	724	213392.6
cpmp27	150	δ	743	54660.5
cpmp32	200	δ	802	82002.0

Table 12. Results of the branch-and-cut on CPMP instances

					Proof		Search		
Name	m	n	type	OPT	C&B	CPX	C&B	CPX	B&P
p20-a	20	40	a	26561	1.0	22.5	1.5	26.7	
p21-a	20	40	a	7295	1.4	8.6	2.1	13.1	
p26-a	20	50	a	4448	1.1	5.3	2.0	12.5	365.2
p30-a	20	50	a	10816	4.1	6260.3	47.7		
p31-a	20	50	a	4466	1.2	4.3	2.4	13.6	2864.4
p33-a	20	50	a	39463	8.8	1056.5	20.3	3061.7	
p36-a	30	60	a	16781	1.7	45.0	2.5	23.3	2008.7
p37-a	30	60	a	14668	3.8	3.9	17.5	12.2	
p39-a	30	60	a	41007	1.0	4.4	6.3	34.3	856.1
p40-a	30	60	a	61633	2.4	44.2	4.2	172.6	
p44-a	30	75	a	36022	3.4	2182.1	7.6		
p46-a	30	75	a	48701	1.5	7.5	3.0	94.4	
p47-a	30	75	a	66230	10.3	87.8	18.8	210.3	
p49-a	30	75	a	79614	3.1	143.3	13.7	790.9	
p51-a	30	90	a	9060	29.4	416.4	94.9	296.4	
p52-a	30	90	a	34652	2.0	27.7	13.0	86.6	
p53-a	30	90	a	30038	1.0	2.5	2.2	14.8	340.0
p55-a	30	90	a	69610	2.6	533.7	3.8	2782.5	
p20-b	20	40	b	13502	0.9	16.3	1.3	34.3	1053.7
p21-b	20	40	b	3870	0.9	6.3	1.2	10.7	400.9
p26-b	20	50	b	2499	1.1	9.9	8.4	17.2	384.0
p30-b	20	50	b	5650	15.0	1239.7	36.6	2697.8	
p33-b	20	50	b	20024	8.0	973.4	26.8	1704.8	
p36-b	30	60	b	8746	1.8	39.5	11.0	112.2	2340.7
p37-b	30	60	b	7681	8.0	10.2	27.7	25.8	
p39-b	30	60	b	20762	1.1	2.5	2.2	16.4	3111.7
p40-b	30	60	b	31047	2.5	41.0	5.9	191.4	
p44-b	30	75	b	18247	3.6	1444.3	6.9		
p46-b	30	75	b	24716	1.8	8.2	2.8	122.6	2676.4
p47-b	30	75	b	33489	16.8	428.2	36.0	1397.0	
p49-b	30	75	b	40104	2.6	136.9	15.2	383.5	
p52-b	30	90	b	17663	2.5	31.2	18.8	104.5	
p53-b	30	90	b	15376	1.1	2.7	1.7	19.3	808.0
p55-b	30	90	b	35162	2.6	663.4	3.6		
p18-d	20	40	d	14418	1.2	6.5	1.2	30.3	555.3
p20-d	20	40	d	24862	0.8	2.9	1.0	11.2	604.1
p33-d	20	50	d	38010	0.7	14.2	1.1	31.9	740.3
p34-d	30	60	d	8323	3.7	4.7	5.2	11.4	188.0
p35-d	30	60	d	8177	2.8	8.2	3.5	39.1	459.9
p42-d	30	75	d	11062	2.7	14.6	40.4	106.3	903.5
p44-d	30	75	d	38049	0.7	3.9	1.8	17.6	211.6
p47-d	30	75	d	67682	1.0	1.8	1.5	15.4	416.9
p48-d	30	75	d	85720	6.2	7.4	21.2	18.6	1408.7
p49-d	30	75	d	92198	1.9	3.1	2.8	14.9	1325.2
p55-d	30	90	d	84207	29.2	599.5	90.4	3670.7	
p57-d	30	90	d	107404	10.3	5.0	11.6	11.0	323.9

Table 13. Results on CNLP instances

partially supported by Russian Foundation for Basic Research, grant 12-07-33045 mol_a_ved.

References

[1] D. Applegate, R. Bixby, V. Chvatal, and W. Cook. Implementing the dantzig-fulkerson-johnson algorithm for large traveling salesman problems. *Mathematical programming*, 97:91–153, 2003.

[2] David L. Applegate, Robert E. Bixby, Vasek Chvatal, and William J. Cook. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton University Press, Princeton, NJ, USA, 2007.

[3] P. Avella, M. Boccia, and I. Vasilyev. A computational study of exact knapsack separation for the generalized assignment problem. *Computational Optimization and Applications*, 45:543–555, 2010.

[4] Pasquale Avella, Maurizio Boccia, and Igor Vasilyev. Computational experience with general cutting planes for the set covering problem. *Operations Research Letters*, 37(1):16 – 20, 2009.

[5] Pasquale Avella, Maurizio Boccia, and Igor Vasilyev. Computational testing of a separation procedure for the knapsack set with a single continuous variable. *INFORMS Journal on Computing*, 24(1):165–171, 2012.

[6] R. Baldacci, E. Hadjiconstantinou, V. Maniezzo, and A. Mingozzi. A new method for solving capacitated location problems based on a set partitioning approach. *Computers and Operations Research*, 29:365–386, 2002.

[7] J.E. Beasley. Or-library: distributing test problems by electronic mail. *Journal of Operational Research Society*, 41(11):1069–1072, 1990.

[8] J.E. Beasley. Or-library: distributing test problems by electronic mail. *Journal of Operational Research Society*, 41(11):1069–1072, 1990.

[9] M. Boccia, A. Sforza, C. Sterle, and I. Vasilyev. A cut and branch approach for the capacitated p-median problem based on fenchel cutting planes. *Journal of Mathematical Modelling and Algorithms*, 7:43–58, 2007.

[10] M.P. Bonami. *Étude et mise en oeuvre dapproches polyédriques pour la résolution de programmes en nombres entiers ou mixtes généraux*. PhD thesis, L'Université Paris 6, 2003.

[11] E.A. Boyd. Generating fenchel cutting planes for knapsack polyhedra. *SIAM Journal of Optimization*, 3:734–750, 1993.

[12] E.A. Boyd. Solving integer programs with cutting planes and preprocessing. *IPCO 1993*, pages 209–220, 1993.

[13] E.A. Boyd. Fenchel cutting planes for integer programming. *Operations Research*, 42:53–64, 1994.

[14] E.A. Boyd. On the convergence of fenchel cutting planes in mixed-integer programming. *SIAM Journal of Optimization*, 5:421–435, 1995.

[15] A. Ceselli. Two exact algorithms for the capacitated p-median problem. *4OR*, 1:319–340, 2003.

[16] A. Ceselli and G. Righini. A branch and price algorithm for the capacitated p-median problem. *Networks*, 45(3):125–142, 2004.

[17] Alberto Ceselli, Federico Liberatore, and Giovanni Righini. A computational evaluation of a general branch-and-price framework for capacitated network location problems. *Annals of Operations Research*, 167:209–251, 2009.

[18] Alberto Ceselli and Giovanni Righini. A branch-and-price algorithm for the multilevel generalized assignment problem. *Operations Research*, 54(6):pp. 1172–1184, 2006.

[19] G. Cornuejols and C. Lemarechal. A convex-analysis perspective on disjunctive cuts. *Mathematical Programming*, 106(3):567–586, 2006.

[20] J. A. Daz and E. Fernndez. A branch-and-price algorithm for the single source capacitated plant location problem. *The Journal of the Operational Research Society*, 53(7):pp. 728–740, 2002.

[21] D. Espinoza. *Étude et mise en oeuvre dapproches polyédriques pour la résolution de programmes en nombres entiers ou mixtes généraux*. PhD thesis, L'Université Paris 6, 2003.

[22] AlanP. French and JohnM. Wilson. Heuristic solution methods for the multilevel generalized assignment problem. *Journal of Heuristics*, 8:143–153, 2002.

[23] Ricardo Fukasawa and Marcos Goycoolea. On the exact separation of mixed integer knapsack cuts. *Math. Program.*, 128(1-2):19–41, 2011.

[24] Fred Glover, John Hultz, and Darwin Klingman. Improved computer-based planning techniques. part ii. *Interfaces*, 9(4):pp. 12–20, 1979.

[25] Kaj Holmberg, Mikael Rnnqvist, and Di Yuan. An exact algorithm for the capacitated facility location problems with single sourcing. *European Journal of Operational Research*, 113(3):544 – 559, 1999.

[26] K. Kaparis and A.N Letchford. Separation algorithms for 0-1 knapsack polytopes. *Mathematical Programming*, 124:69–91, 2010.

[27] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2005.

[28] Manuel Laguna, James P. Kelly, JosLuis Gonzlez-Velarde, and Fred Glover. Tabu search for the multilevel generalized assignment problem. *European Journal of Operational Research*, 82(1):176 – 189, 1995.

[29] L. Lorena and E. Senne. A column generation approach to capacitated p-median problems. *Computers and Operations Research*, 31(6):863–876, 2004.

[30] S. Martello and P. Toth. *Knapsack Problems - Algorithms and Computer Implementations*. Wiley, 1990.

[31] Mara A. Osorio and Manuel Laguna. Logic cuts for multilevel generalized assignment problems. *European Journal of Operational Research*, 151(1):238 – 246, 2003.

[32] A. Pigatti, M. Poggi de Aragao, and E. Uchoa. Stabilized branch-and-cut-and-price for the generalized assignment problem. In *2nd Brazilian Symposium on Graphs, Algorithms and Combinatorics, Electronic Notes in Discrete Mathematics, Vol. 19*, pages 389–395, 2005.

[33] A. Pigatti, M. Poggi de Aragao, and E. Uchoa. Stabilized branch-and-cut-and-price for the generalized assignment problem. In *2nd Brazilian Symposium on Graphs, Algorithms and Combinatorics, Electronic Notes in Discrete Mathematics, Vol. 19*, pages 385–395, 2005.

[34] D. Pisinger. A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 46(0):758–767, 1995.

[35] Marius Posta, JacquesA. Ferland, and Philippe Michelon. An exact method with variable fixing for solving the generalized assignment problem. *Computational Optimization and Applications*, 52:629–644, 2012.

[36] I. Vasilev. A cutting plane method for knapsack polytope. *Journal of Computer and Systems Sciences International*, 48:70–77, 2009.