Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

Cutting plane algorithms for 0-1 programming based on cardinality cuts

Osman Oguz *

Department of Industrial Engineering, Bilkent University, Ankara, Turkey

A R T I C L E I N F O

Article history:
Received 6 November 2008
Accepted 2 January 2010
Available online 25 January 2010

Keywords:
0-1 Integer programming
Valid inequalities
Cutting planes

A B S T R A C T

We present new valid inequalities for 0-1 programming problems that work in similar ways to well known cover inequalities. Discussion and analysis of these cuts is followed by their revision and use in integer programming as a new generation of cuts that excludes not only portions of polyhedra containing noninteger points, also parts with some integer points that have been explored in search of an optimal solution. Our computational experimentations demonstrate that this new approach has significant potential for solving large scale integer programming problems.

1. Introduction

Branch-and-cut was introduced in [2] demonstrating the important role of the use of Gomory cutting planes [4] and cover inequalities in the branch-and-bound process for solving integer programming problems. Relatively recent works like [5,6] provide extensive discussions of available strategic choices for using cover inequalities in the branch-and-cut process for 0-1 programming. One may see [7,10] for basic expositions of the subject and related issues.

We work on the 0-1 integer programming problem given below to introduce new valid inequalities similar to cover and lifted cover inequalities. We have chosen this problem to introduce our approach, because most of the work on cover inequalities are based on this problem. As a good example, we can mention [3] that describes an implementation of cover cuts on the multiple knapsack version of the problem.

$$\textbf{IP}\quad \text{Maximize}\quad z=\sum_{j=1}^{n}c_jx_j \tag{1}$$

$$\text{subject to}\quad \sum_{j=1}^{n}a_{ij}x_j\leqslant b_i\quad \text{for } i=1,\ldots,m, \tag{2}$$

$$x_j\in\{0,1\}\quad \text{for } j=1,\ldots,n, \tag{3}$$

m and n are the number of constraints and decision variables, respectively.

We do not assume any restrictions the integrality or nonintegrality of the parameters c_j, a_{ij} and b_i.

The next section consists of the description and the generation method of the inequalities together with the proof of validity. Section 3 is devoted to redefining and improving the performance of the proposed cuts. The preliminary numerical experiments are discussed in Section 4. Conclusions and comments follow in Section 5.

2. The new cut

Consider the problem **IP** and let $X_{\mathrm{LP}}=(x_1,\ldots,x_n)$ denote a solution to the linear programming (LP) relaxation of this problem. Also let $S_p=\{j|x_j\ >0 \text{ in } X_{\mathrm{LP}}\}$ and solve the following problem:

$$z_0=\max\sum_{j=1}^{n}x_j:\sum_{j=1}^{n}a_{ij}x_j\leqslant b_i\quad\text{for } i=1,\ldots,m\quad\text{and } x_j \in[0,1]\quad \text{for } j=1,\ldots,n. \tag{4}$$

The following inequality is obviously valid:

$$\sum_{j\in S_p}x_j\leqslant z_0. \tag{5}$$

Also, this inequality is valid for all possible values x_j in any solution of the LP relaxation for any objective function. Moreover, the inequality is valid in the form,

$$\sum_{j\in S_p}x_j\leqslant\lfloor z_0\rfloor \tag{6}$$

for any integer solution of the problem for any objective function. In fact, this last inequality may be an effective cut to make some noninteger solutions infeasible. However, its use can be limited to very few instances and it becomes ineffective very soon in a cutting plane framework. It may even be useless if z_0 is integer valued or $\sum_{j\in S_p}x_j\leqslant\lfloor z_0\rfloor$ is not violated by the relaxed solution. Nonetheless, it is the starting point of our proposal for a new type of cuts.

* Tel.: +90 312 290 1544; fax: +90 312 266 4054.
E-mail address: ooguz@bilkent.edu.tr

Starting with the solution to the LP relaxation of the problem **IP**, we partition N, the index set of variables of **IP** into two subsets: $S_1 = \{j x_j = 1 \text{ in } X_{\text{LP}}\}$ and $S_2 = \text{N} \setminus S_1$. Then, naming the LP relaxation of the original integer program **IP** as **P**, we define the following linear program named as **P1**:

$$\textbf{P1}\quad z_1 = \text{maximize} \sum_{j\in S_2} x_j \tag{7}$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij}x_j \leqslant b_i \quad \text{for } i = 1,\dots,m, \tag{8}$$

$$\sum_{j\in S_1} x_j = |S_1|, \tag{9}$$

$$0 \leqslant x_j \leqslant 1 \quad \text{for } j = 1,\dots,n. \tag{10}$$

Then the following is true.

Proposition 1. *The inequality*

$$\sum_{j\in S_1} rx_j + \sum_{j\in S_2} x_j \leqslant r|S_1| + \lfloor z_1 \rfloor \tag{11}$$

is valid for the solution set of the problem **IP** *for* $r = |S_2|$.

Proof. It is obvious that the above inequality is violated by the current X_{LP} if $\sum_{j\in S_2} x_j > \lfloor z_1 \rfloor$. On the other hand, the inequality is valid for all integer solutions satisfying the condition $\sum_{j\in S_2} x_j \leqslant \lfloor z_1 \rfloor$. However, when $\sum_{j\in S_2} x_j > \lfloor z_1 \rfloor$ holds for an integer solution, that solution must have at least one x_j with $j \in S_1$ equal to zero in order that the solution is feasible, by the definition of z_1 in **P1**. Thus we conclude that the number of x_j's for $j \in S_2$ being equal to 1 can be greater than $\lfloor z_1 \rfloor$, only if at least one variable x_j in S_1 is equal to 0. Setting $r = |S_2|$ will allow the inequality to hold even when all variables in S_2 are equal to 1, and only one variable in S_1 is equal to 0. □

The valid inequality of Eq. (11) will be called the cardinality cut.

Assigning large values to the parameter ***r*** is not desirable since the quality of the cut will be poor, i.e., the cut will remove a relatively small part of the underlying polyhedron containing no integer solutions. A more reasonable approach is to choose ***r*** more carefully. Consider a slight variation of **P1** given below:

$$\textbf{P2}\quad z_2 = \text{maximize} \sum_{j\in S_2} x_j \tag{12}$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij}x_j \leqslant b_i \quad \text{for } i = 1,\dots,m, \tag{13}$$

$$\sum_{j\in S_1} x_j = |S_1| - 1, \tag{14}$$

$$0 \leqslant x_j \leqslant 1 \quad \text{for } j = 1,\dots,n. \tag{15}$$

Assuming $z_2 > z_1$, without loss of generality, $z_2 - \lfloor z_1 \rfloor$ is an upperbound on how many more x_j's in S_2 can take a value of 1 when we decrease the cardinality of S_1 by 1. Note that when we replace the right hand side of the equation $\sum_{j\in S_1} x_j = |S_1| - 1$ by $|S_1| - 2$, the difference $z_2 - \lfloor z_1 \rfloor$ will less than double, since z_2 parametrized by $|S_1| - k$, is a concave piecewise linear function for $k \geqslant 0$. Thus, setting $r = z_2 - \lfloor z_1 \rfloor$ gives a relaxation sufficiently tight for the purpose of cutting deeper into the underlying polyhedron.

There will, of course, be instances ($z_1 = \lfloor z_1 \rfloor$ for example), such that the valid inequality will fail to eliminate X_{LP}. We can try a few more things before giving up and starting branching. The most obvious thing to do is to play around the partition of N into S_1 and S_2. We have tried two strategies with partial success. First one is to move few variables from S_2 to S_1 picking those with values close to 1. Second strategy is to eliminate some variables in S_2 from consideration, i.e., not including them in the valid inequalities, or in the objective function of problem **P2**. We may end up with effective cuts as a result of these changes. Second strategy and its variations seem to be working better in our preliminary experimentations.

We report results comparing the efficiency of these new cuts with that of the cover inequalities in a cutting plane framework on a set of hard multidimensional knapsack problems described in [8,9]. Note that, the new cuts may be used for the traveling salesman problem, set packing or covering problems, and other **NP-*Hard*** problems with 0-1 constraint matrices as well, without any adaptation of the inequality given in Eq. (11). This is an extra feature of the new cut over the capability of ordinary cover inequalities.

Although the comparison mentioned above indicates superior efficiency of the cardinality cuts over the cover inequalities, we have discovered that their functionality and efficiency may be further enhanced by slightly changing their definition and using them in a novel algorithmic approach. This led to some significant improvements for the solution of large scale 0-1 integer programming problems. The next section reports these developments.

3. Redefinition, optimization and aggregation of the cardinality cuts

The inequality given by Eq. (11) is a lifted version of the inequality of Eq. (6). We take a further step in this direction and obtain what might be called the overlifted version of Eq. (11), because lifting is done for the purpose of eliminating a certain set of feasible integer solutions from the solution space.

Let us consider the following revised version of **P1** defined by Eqs. (7)–(10):

$$\textbf{P3}\quad z_3 = \text{maximize} \sum_{j\in S_2} c_j x_j \tag{16}$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij}x_j \leqslant b_i \quad \text{for } i = 1,\dots,m, \tag{17}$$

$$\sum_{j\in S_1} x_j = |S_1|, \tag{18}$$

$$x_j \in \{0,1\} \quad \text{for } j = 1,\dots,n. \tag{19}$$

The optimal solution of this problem is a feasible solution for **IP**. Also, $\text{ZINT}_{\text{LB}} = z_3 + \sum_{j\in S_1} c_j$ is a lower bound for the optmal objective function value of **IP**. We call the following version of the cardinality cut "the optimized cardinality cut":

$$\sum_{j\in S_1} rx_j + \sum_{j\in S_2} x_j \leqslant r|S_1| \tag{20}$$

with $r = |S_2|$.

Then, we state and prove the following proposition:

Proposition 2. *The optimized cardinality cut represented by the inequality in Eq. (20), when added to* **P**, *makes sure that all solutions of* **P3**, *except for the solution* $X = \{j|x_j = 1 \text{ for } j \in S_1 \text{ and } x_j = 0 \text{ for } j \in S_2\}$, *are infeasible while all other integer solutions of* **IP** *remain feasible.*

Proof. It is obvious that for any positive value of r, the inequality will be violated when any variable x_j for $j \in S_2$ takes a positive value while all $x_j = 1$ for $j \in S_1$. On the other hand, setting only one $x_j = 0$ such that $j \in S_1$ will make room for all x_j's such that $j \in S_2$to take positive values if we set $r = |S_2|$. □

Corollary 1. *When the inequality of Eq. (20) is added to the constraint set of* **P**, *the LP relaxation of* **IP**, *with* $r = |S_2|$, *it separates only the integer and noninteger solutions having all* $x_j = 1$ *with* $j \in S_1$ *as a proper subset of variables with nonzero values, from the feasible set of* **P**. *All other integer feasible solutions remain in the feasible set.*

Note that the cut represented by Eq. (20) is not a conventional polyhedral cut aimed at defining the convex hull of the integer solutions of our problem. It cuts into the polyhedron representing the convex hull, eliminating fathomed integer solutions only.

Obviously, we can use this inequality as a cut in the constraint set of problem **P**, as long as we keep the integer solution corresponding to $\mathrm{ZINT}_{\mathrm{LB}}$ as an incumbent until it is replaced with a better one, i.e., a solution that gives a higher lower bound. Again, to enhance the effectiveness of the cut, we can set $r = z_2$, where z_2 is obtained from Eq. (12) in the solution of **P2**.

Obviously, the optimized version of the cardinality cut is more effective than the original version due to the fact that the original version is ineffective when $\sum_{j\in S_2} x_j \leqslant \lfloor z_1 \rfloor$ holds, in which case the related inequality cannot separate the noninteger solution from the solution set of **P**, whereas the new version is always able to do the separation. Of course, there is a cost: we have to solve the integer programming problem **P3**, to be able to use the inequality given in Eq. (20) as a cut. Considering the cardinality of S_2, the number of variables of the integer programming in question (**P3**) may be quite large, and solving **P3** to optimality may be very time consuming. In fact, the use of optimized cardinality cuts may be impossible if the size of **P3** is too large, in which case we can use the cut only in its original form in Eq. (11).

To reduce the number of variables and the solution time of **P3**, we fix the variables with a value equal to 0 in the LP relaxation of **P**, just like the variables in S_1. Fixing is achieved by revising the constraint in Eq. (18) of **P3** as follows:

Let $S_2 = S_f \cup S_0$, where $S_f = \{j | 0 < x_j < 1, j \in S_2\}$, and $S_0 = \{j | x_j = 0, j \in S_2\}$. Then the equality given in Eq. (18) takes the following form:

$$\sum_{j\in S_1} x_j - \sum_{j\in S_0} x_j = |S_1|. \tag{21}$$

After solving **P3** with this constraint, the inequality of the optimized cardinality cut given in Eq. (20) becomes:

$$\sum_{j\in S_1} r x_j - \sum_{j\in S_0} r x_j + \sum_{j\in S_f} x_j \leqslant r|S_1|, \tag{22}$$

where the objective function value z_2 in Eq. (12) is redefined as $z_2 = \sum_{j\in S_f} x_j$ and Eq. (14) is replaced by

$$\sum_{j\in S_1} x_j - \sum_{j\in S_0} x_j = |S_1| - 1 \text{ in } \mathbf{P2}.$$

The solution time of **P3** is reduced as expected by using Eq. (21), however there is a drawback: the effectiveness of Eq. (22) as a cut is much less relative to that of Eq. (20). Thus, when we use it in a cutting plane framework, the result is a slower improvement (decrease) of the upper bound and adding a greater number of cuts (constraints) leading to an increased number of fractional variables in the LP relaxation that defeats the purpose of reducing the number of variables for **P3**.

This led to a further refinement of the approach and use of an unconventional way of aggregation of these cuts. Consider two unidentical cuts; Cut1 and Cut2 represented by the two inequalities given below:

$$\text{Cut1}: \quad \sum_{j\in S_1^1} r_1 x_j - \sum_{j\in S_0^1} r_1 x_j + \sum_{j\in S_f^1} x_j \leqslant r_1 \left|S_1^1\right|. \tag{23}$$

$$\text{Cut2}: \quad \sum_{j\in S_1^2} r_2 x_j - \sum_{j\in S_0^2} r_2 x_j + \sum_{j\in S_f^2} x_j \leqslant r_2 \left|S_1^2\right| \tag{24}$$

and Cut3 which is a combination of Cut1 and Cut2 as described below:

$$\text{Cut3}: \quad \sum_{j\in S_1^3} r_3 x_j - \sum_{j\in S_0^3} r_3 x_j + \sum_{j\in S_f^3} x_j \leqslant r_3 \left|S_1^3\right|, \tag{25}$$

where:

$$S_1^3 = S_1^1 \cap S_1^2, \quad S_0^3 = S_0^1 \cap S_0^2 \quad \text{and } S_f^3 = \mathrm{N} \setminus \{S_1^3 \cup S_0^3\}. \tag{26}$$

We are now ready to state the following proposition:

Proposition 3. *Let* $z(1\&2)$ *represent the value of the objective function of* **P** *solved with Cut1 and Cut2 added as constraints, and* $z(3)$ *the objective function of the same problem solved with Cut3 only. Then,* $z(3) \leqslant z(1\&2)$ *holds.*

Proof. Let T_1 be the set of feasible solutions of **P** with $x_j = 1$ for $j \in S_1^1, x_j = 0$ for $j \in S_0^1$, and $0 < x_j < 1$ for $j \in S_f^1$. T_2 and T_3 are similarly defined using the corresponding sets associated with Cut2 and Cut3. Then we have $\{T_1 \cup T_2\} \subset T_3$, due to the fact that $S_1^3 \subset S_1^1$ and $S_1^3 \subset S_1^2, S_0^3 \subset S_0^1$ and $S_0^3 \subset S_0^2$, and also $\{S_f^1 \cup S_f^2\} \subset S_f^3$ are true by definition. Let T represent the set of feasible solutions of **P**. The following is true:

$T \setminus T_3 \subset T \setminus \{T_1 \cup T_2\}$ because $\{T_1 \cup T_2\} \subset T_3$ is true, which implies $z(3) \leqslant z(1\&2)$ □

Proposition 3 enables us to throw away Cut1 and Cut2, and solve **P** with only Cut3 added, and obtain a better upperbound as a result. Note that the aggregation can be repeated recursively for any number of times as long as S_0 and S_1 are not both empty. This gives us the opportunity to solve **P**, define Cut1, then add Cut1 to **P** and re-solve, define Cut2, aggregate Cut1 and Cut2 to obtain Cut3, then replace Cut1 by Cut3, and re-solve **P**, define Cut4 and aggregate it with Cut3 to obtain Cut5 which replaces Cut3, and so on until defining Cut(K), at which point we solve **P3** to obtain an integer incumbent solution. Once this is done, Cut(K) becomes a permanent constraint of **P**, we restart the process by solving **P** and defining Cut1 again. The basic advantage of this approach is to keep the number of added cuts low while obtaining significant reductions of the upperbound and keeping the number of variables of **P** under control. This approach seems to have significant potential value and will be discussed in more detail, including a formal algorithmic description, in Section 4.

4. Computational experiments

This section is divided into three subsections. In the first one, the performance of the initial cardinality cut given in Eq. (11) is compared with that of cover inequalities on well known hard knapsack problems in the literature in a cutting plane framework. The second subsection is devoted to an implementation of optimized cardinality cuts in the form given in Eq. (20) on TSP problems taken from TSPLIB (http://www.iwr.uni-heidelberg.de/groups/comopt/soft/TSPLIB95/TSPLIB.html). The final subsection is about using the aggregation of optimized cardinality cuts on some large scale multidimensional 0-1 knapsack problems.

4.1. Comparison of cardinality cuts with cover inequalities:

We have used a subset of the multidimensional 0-1 knapsack test problems provided in [8]. These problems are inherently difficult, much like but not exactly the same with the market share problems presented in [1] .The table below displays the results for the set of first 5 problems from each of the first 9 categories given in OR-Library, available at (http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapinfo.html).

The experiments have been carried out on a 2.6 GHz, 2xAMD Opteron server running Linux with 2 GB RAM. The generation of the cardinality cuts has been embedded into the branch-and-cut framework of CPLEX 8.0.1(*Interactive Optimizer* 8.1.0, Copyright (c) ILOG 1997–2002). We report the results associated with the root node of the branch-and-cut tree only. In the table below, columns under CPLEX give performance data for cover inequalities, and columns under CC give the data related to cardinality cuts. INITIAL is the objective function value of the LP relaxation, and "improved" is the same bound after addition of cuts. "improvement" is equal to INITIAL – "improved". Columns Cover Cuts and Cuts display the number of cuts generated by the two methods. A lower bound obtained for each time **P1** is solved by computing $\sum_{j\in S_1} c_j x_j + \sum_{j\in S_2} c_j \lfloor x_j \rfloor$, and LB is the best of these values. GAP is equal to (improvement)/(INITIAL- LB).

The time and bound improvement performance of CPLEX at the root node, reflects the results obtained when only cover inequalities are allowed as cutting planes. All other cuts, including Gomory fractional cuts are disabled.

Note that, in many cases CPLEX is unable to find a cover inequality, but still reports slight improvements, i.e., positive (INITIAL – "improved"). Apparently, CPLEX employs some heuristics at the root node to find an initial lower bound which improves the upper bound in the process. Being unable to shut off these heuristics, we present our results with this bias on the side of the performance of CPLEX. Time(s) means the amount of time spent for separation.

We have devised a simple greedy heuristic to find effective cardinality cuts. It stops when it cannot separate the current solution (LP relaxation) from the solution space of **P**. The results in the CC columns in Table 1 were obtained by using this algorithm.

The Heuristic algorithm:

Step 1: Solve **P** and define S_1 and $S_2 = \{S_f \cup S_0\}$. If $S_f = \phi$, stop. The solution is integer optimal. Otherwise let x_j^1 for $j \in S_f$ denote the value of the jth variable in the LP relaxation.
Step 2: Solve **P1**. Let x_j^2 for $j \in S_f$ denote the value of the jth variable in this solution.
Step 3: Compute $f_1 = \sum_{j\in S_f} x_j^1$ and $f_2 = \sum_{j\in S_f} x_j^2$If $\lfloor f_2 \rfloor < f_1$, add a new cardinality cut to **P** with $r = z_2 - \lfloor z_1 \rfloor$, and go to Step 1. Otherwise go to Step 4.
Step 4: If $S_f = \phi$, stop. Output current upper bound and lower bound, if any exists. Otherwise, find j^* such that $\frac{x_{j^*}^1}{x_{j^*}^2}$ is smallest for $j \in S_f$. Set $S_f = S_f \setminus (j^*)$ and go to Step 2.

In the results obtained for the cardinality cuts (CC), CPLEX is used as a LP solver only.

The summary given in Table 2 shows the extent of the gap improvement the cardinality cuts can provide in less than a second for the problems in the test set in Table 1:

4.2. Some results with TSP problems

We have also solved a few problems from the library of TSP problems from the following web page: TSPLIB (http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/) in our experiments.

The algorithm uses the optimized cardinality cut given in Eq. (20).

We have implemented the following simple algorithm:

Step 0. Set INC = +∞ as the value of the incumbent tour.
Step 1. Solve **P** (2-matching formulation). If the solution is a tour or if the value of the objective function of a**P** is greater than INC, stop. Otherwise, define S_1 and S_2.
Step 2. Set $r = n - |S_1|$. Solve **P3**. If the solution is integer, but represents subtours, then add a subtour breaking constraint to **P** and go to step 1. If the solution is a tour, record it as an incumbent and revise INC. If the current solution is better than the current incumbent, add a tour breaking constraint to **P** eliminating the tour and go to Step 1. When **P3** is infeasible (no integer solution exists) add the cardinality cut given in Eq. (20) to **P** and go to Step 1.

The solutions to **P** and **P3** were obtained using CPLEX version 8.0.1 on the same server. The choice of $r = n - |S_1|$ is specific to TSP, because n, the number cities, gives a natural upperbound on the number of variables that can be equal to 1 in any solution.

Our attempts to solve larger problems were not successful due to excessive solution times for the subproblem **P3**. The results reported in the table may seem unimpressive in comparison to computation times reported in Concorde TSP Solver webpage (http://www.tsp.gatech.edu/concorde/index.html). One should keep in mind that we have used CPLEX Mip Solver supplemented by subtour breaking constraints and cardinality cuts only. The version of the cardinality cut (Eq. (20)) is not the most potent cut proposed (i.e., the aggregated version is the most potent) in this paper. LP relaxations are solved from scratch each time a cut is added to the problem. That is, neither the algorithm and nor the computer program we used are state of the art caliber. Our purpose is to demonstrate that the proposed cuts works for problems other than the multidimentional 0-1 knapsack problem, a claim made in Section 2 of this paper.

4.3. Experiments with the multidimentional Knapsack problems

The cutting plane algorithm used to solve the multidimentional knapsack problems in this subsection uses the aggregated cardinality cut given in Eq. (25). An outline of the algorithm based on agregated cardinality cuts is as follows:

Step 0. Set INC $= -\infty$ as the value of the incumbent solution, k=0, and assign a value to the aggregation frequency $1 \leqslant K \leqslant 1000$.
Step 1. Solve **P** (the LP relaxation). If the solution is integer or if the value of the objective function of **P** is smaller than INC, stop and declare INC as the optimal value. Otherwise, define S_1, S_2 and S_f.
Step 2. Solve the problem:

$$\text{Max} \quad r = \sum_{j\in S_f} x_j \text{ st} : \sum_{j=1}^{n} a_{ij} x_j \leqslant b_i \quad \forall i = 1,\ldots,m, \sum_{j\in S_1} x_j - \sum_{j\in S_0} x_j = |S_1| - 1 \quad \text{and } 0 \leqslant x_j \leqslant 1 \quad \forall j = 1,\ldots,n,$$

to determine the value of $\boldsymbol{r}$ for the current partition of N into S_1^k, S_2^k and S_f^k. If $k = 0$, define a cardinality cut with the current $\boldsymbol{r}$ and current partition. Add this cut to **P**, set $k = k + 1$ and go to step 1. If $0 < k < K$, go to the next step. Otherwise go to Step 4.

Step 3. Carry out an aggregation procedure as described by Eqs. (23)–(25) to combine S_1^{k-1}, S_2^{k-1}

and S_f^{k-1} with S_1^k, S_2^k and S_f^k, calling the result the kth partition. Determine $\boldsymbol{r}$ with the kth partition and replace the last added cut to **P** by the new cut obtained using the kth partition and go to Step 1.

Step 4. Carry out an aggregation procedure combining $K - 1$st partition with the Kth (the result is still called the Kth partition), and solve **P3**. If the solution is better than the incumbent, record it as the new incumbent. Replace the $K - 1$st cut by the Kth in **P**, set $k = 0$ and go to Step 1.

Table 1
Results for ORLIB problems with cardinality cuts and cover cuts.

	INITIAL	CPLEX			CC				Improvement		**GAP** % closed	
		Improved	Time (s)	Cover cuts	Improved	Time (s)	Card. cuts	LB	CPLEX	CC	CPLEX	CC
mk1_1	24585.90	24583.33	0.00	2	24577.90	0.02	19	20810.00	2.57	8.00	0.0681	0.2119
mk1_2	24538.21	24537.25	0.00	1	24512.20	0.12	75	19799.00	0.96	26.01	0.0203	0.5488
mk1_3	23895.83	23894.83	0.00	0	23887.11	0.02	28	21254.00	1	8.72	0.0379	0.3301
mk1_4	23724.14	23718.54	0.00	1	23717.16	0.02	29	19984.00	5.6	6.98	0.1497	0.1866
mk1_5	24223.03	24208.54	0.00	3	24223.03	0.00	0	21859.00	14.49	0.00	0.6129	0.0000
mk2_1	59442.47	59439.95	0.01	2	59438.76	0.01	4	55919.00	2.52	3.71	0.0715	0.1053
mk2_2	61629.34	61627.71	0.00	2	61627.45	0.01	3	58131.00	1.63	1.89	0.0466	0.0540
mk2_3	62259.54	62254.77	0.01	3	62256.51	0.03	33	58801.00	4.77	3.03	0.1379	0.0876
mk2_4	59578.23	59574.75	0.00	2	59575.64	0.04	33	56328.00	3.48	2.59	0.1071	0.0797
mk2_5	59078.37	59077.73	0.01	2	59075.18	0.04	36	54437.00	0.64	3.19	0.0138	0.0687
mk3_1	120234.92	120233.59	0.03	2	120231.82	0.07	25	116251.00	1.33	3.10	0.0334	0.0778
mk3_2	117955.16	117954.19	0.03	1	117952.31	0.07	36	113953.00	0.97	2.85	0.0242	0.0712
mk3_3	121213.33	121211.51	0.03	1	121211.65	0.01	4	118497.00	1.82	1.68	0.0670	0.0618
mk3_4	120888.52	120887.53	0.03	1	120885.15	0.02	9	117692.00	0.99	3.37	0.0310	0.1054
mk3_5	122426.49	122425.49	0.03	0	122423.55	0.05	21	119430.00	1	2.94	0.0334	0.0981
mk4_1	23480.64	23479.75	0.01	1	23475.36	0.02	15	19067.00	0.89	5.28	0.0202	0.1196
mk4_2	23220.69	23218.94	0.00	1	23189.08	0.06	37	15735.00	1.75	31.61	0.0234	0.4223
mk4_3	22493.74	22493.42	0.01	0	22462.74	0.03	37	16800.00	0.32	31.00	0.0056	0.5445
mk4_4	23087.47	23086.91	0.00	0	23085.14	0.01	3	18701.00	0.56	2.33	0.0128	0.0531
mk4_5	23073.88	23067.04	0.01	0	23073.88	0.00	0	19765.00	6.84	0.00	0.2067	0.0000
mk5_1	59489.34	59487.98	0.01	0	59489.34	0.01	0	56566.00	1.36	0.00	0.0465	0.0000
mk5_2	59024.30	59019.15	0.01	0	59018.56	0.09	55	53594.00	5.15	5.74	0.0948	0.1057
mk5_3	58413.15	58408.88	0.01	0	58410.50	0.00	1	53918.00	4.27	2.65	0.0950	0.0590
mk5_4	61263.00	61262.64	0.01	0	61259.40	0.03	15	55742.00	0.36	3.60	0.0065	0.0652
mk5_5	58363.34	58363.27	0.01	0	58360.96	0.04	28	53457.00	0.07	2.38	0.0014	0.0485
mk6_1	118019.48	118019.36	0.03	0	118018.63	0.03	14	112120.00	0.12	0.85	0.0020	0.0144
mk6_2	119437.29	119435.40	0.03	0	119434.31	0.14	53	114659.00	1.89	2.98	0.0396	0.0624
mk6_3	119405.70	119405.22	0.03	0	119404.16	0.04	16	113140.00	0.48	1.54	0.0077	0.0246
mk6_4	119066.09	119065.33	0.03	0	119065.77	0.02	2	115583.00	0.76	0.32	0.0218	0.0092
mk6_5	116697.95	116696.90	0.02	0	116695.34	0.72	118	108457.00	1.05	2.61	0.0127	0.0317
mk7_1	22579.07	22577.01	0.01	0	22569.71	0.03	4	13406.00	2.06	9.36	0.0225	0.1020
mk7_2	22367.84	22366.69	0.01	0	22340.29	0.12	37	12226.00	1.15	27.55	0.0113	0.2716
mk7_3	21270.50	21270.18	0.01	0	21270.50	0.01	0	15097.00	0.32	0.00	0.0052	0.0000
mk7_4	22049.63	22048.72	0.01	0	22041.90	0.09	43	13416.00	0.91	7.73	0.0105	0.0895
mk7_5	22529.25	22528.11	0.01	0	22524.12	0.05	21	13512.00	1.14	5.13	0.0002	0.0006
mk8_1	57430.15	57424.93	0.05	0	57418.76	0.15	25	45782.00	5.22	11.39	0.0448	0.0978
mk8_2	59080.03	59079.42	0.04	0	59072.10	0.07	8	47886.00	0.61	7.93	0.0054	0.0708
mk8_3	57176.74	57173.74	0.04	0	57174.91	0.08	10	49387.00	3	1.83	0.0385	0.0235
mk8_4	57568.92	57568.16	0.04	0	57568.92	0.01	0	49671.00	0.76	0.00	0.0096	0.0000
mk8_5	57277.70	57276.06	0.04	0	57273.85	0.03	1	47251.00	1.64	3.85	0.0164	0.0384
mk9_1	116619.01	116618.56	0.09	0	116615.91	0.24	22	100956.00	0.45	3.10	0.0029	0.0198
mk9_2	115370.13	115368.78	0.08	0	115365.20	0.14	9	105705.00	1.35	4.93	0.0140	0.0510
mk9_3	117342.45	117341.16	0.10	0	117339.31	0.13	7	105799.00	1.29	3.14	0.0112	0.0272
mk9_4	115946.40	115944.56	0.09	0	115943.20	0.24	20	103239.00	1.84	3.20	0.0145	0.0252
mk9_5	117079.29	117079.01	0.08	0	117077.99	0.68	54	104622.00	0.28	1.30	0.0022	0.0104

Table 2
Summary results in terms of relative improvements.

76.67%	Percentage better
29.05%	Percentage 5 times better
17.62%	Percentage 10 times better
9.05%	Percentage 20 times better

Table 3
Solution times for some TSPLIB problems.

Problem name	CPU seconds	Number of cardinality cuts added
Berlin52.tsp	5.20	0
Eil51.tsp	50.34	65
Eil76.tsp	271.13	8
Eil101.tsp	231.78	189
Gr96.tsp	605.14	65
Bier127.tsp	2369.62	422
Ch130.tsp	1866.62	338
Ch150.tsp	5252.53	413

Implementation of this algorithm on a subset of problems given in [8] gave the results listed in Table 4 below.

The numbers in the columns of Cardinality Cuts are the times in *cpu* seconds used for solving the corresponding knapsack problems with the algorithm given above. The column for CPLEX gives the times for the same problems by using CPLEX alone.We have used $K = 30, 50, 50$ for problems with 100, 250 and 500 variables correspondingly. CPLEX tolerance level (mipgap) was set equal to zero for solution times in both columns.

The version used is CPLEX (*Interactive Optimizer* 8.1.0 , Copyright (c) ILOG 1997–2002).

These problems are specially designed to be difficult to solve by branch and cut methods. Correlation between the objective function and the constraints, and the tightness of the constraints are the basic design parameters. These parameters have different settings to create problems with varying difficulty. For more details, the web address given above may be seen.

In two cases (*mknapcb3_30, mknapcb5_1*) CPLEX stopped giving "*out of memory*" message without identifying the optimal solution.

The next set of experiments are carried out on relatively larger problems we have generated using our own pseudo random number generator. Multidimentional 0-1 knapsack problems were generated in the range of 1000–20,000 variables and 25–100

Table 4
Solution times in CPU seconds for some ORLIB problems.

Problem name	Cardinality cuts *Cpu seconds*	CPLEX *Cpu seconds*
Mknapcb1_1	0.31	0.32
Mknapcb1_15	3.00	2.38
Mknapcb1_30	0.06	0.05
Mknapcb2_1	19.20	26.62
Mknapcb2_15	707.18	771.15
Mknapcb2_30	92.88	109.81
Mknapcb3_1	11700.26	12590.24
Mknapcb3_15	861.89	949.26
Mknapcb3_30	29067.77	>3626.86
Mknapcb4_1	132.98	104.13
Mknapcb4_15	11.68	10.36
Mknapcb4_30	3.95	4.87
Mknapcb5_1	16298.55	>4684.9
Mknapcb5_15	13149.29	13459.81
Mknapcb5_30	2825.75	3048.88

constraints. The objective function coefficients are uniformly distributed integers between 0 and 500, the constraint coefficients are also uniform random numbers between 0 and 100. Right hand side constants are obtained by multiplying the sum of the constraint coefficients corresponding to the constraint in question multiplied by a uniform random number between 0.1 and 0.90, and rounded to the nearest integer. Table 5 below displays the time performance of the algorithm given at the begining of this subsection together with time performance of CPLEX on twelve relatively large problems generated as described above. The first column of the table shows the dimensions of problems. Second column is the *cpu* times for the algorithm that uses aggregated cardinality cuts, and the third column that of CPLEX. CPLEX stopped without finding the optimal solution with a "out of memory" message for the problems marked with stars in the table. We set $K = 100$ as default. This value was increased for three problems to speed up the coverage of the optimality gap, i.e., the difference between the incumbent value and the available upperbound. Algorithm using cardinality cuts were performed worse than CPLEX only in two cases: problems 25×1000 and $25 \times 20,000$. In both cases the difference between value of the optimal solution and the objectve function value of the LP relaxation were very small, less than one in case of problem $25 \times 20,000$. The setup time for a single cardinality cut required for solving the LP relaxation K times, probably took much longer than CPLEX to cover the small optimality gap for these problems.

The solution time records in the table speak for themselves. Six of the twelve problems are probably solvable by the use of cardinality cuts only. The number of cuts used to achieve these results may also be considered very reasonable.

5. Conclusions and comments

In this paper, the description and the proofs of validity of cardinality cuts are given in three stages. The first stage introduces the cut in its most crude form that may be considered as an integer rounding procedure. The computational experiments presented in Section 4 provide evidence that these cuts may be more effective than cover cuts: one of the main forces in "cut" part of the branch-and-cut routines.

The second stage is the optimization of cardinality cuts through integer programming. Solving relatively small subproblems by integer programming allowed the definition and use of a new generation of cuts that can cut off parts of the underlying polyhedron containing integer solutions that are fathomed. These cuts were used to solve a small set of TSP test problems from the literature. The results displayed in Table 3 seem promissing to say the least.

The third stage describes the development of the aggregated cardinality cut that can be used to solve considerably large problems by means of solving relatively small subproblems a few times by integer programming and adding an aggregated cut to the main problem after solving each subproblem. Most of the problems in Table 5, for example, were solved by solving a 0-1 integer programs with a couple of hundred variables only and by adding a single cut to the original problem. However, this does not mean that higher numbers are unlikely.

Main feature of the approach is the use of the information obtained from the solution of the subproblem **P3**. The solution provides a potential incumbent and a cardinality cut that separates all solutions of this subproblem from the polyhedron representing the solution space of the original problem. The speed of reduction of the upper bound on the value of the optimal solution is remarkable especially in the case of aggregated version of the cuts. Optimized cardinality cuts are vastly superior to cardinality cuts (Eq. (11)). None of the TSP problems of Table 3 could be solved using cardinality cuts only. Similarly, aggregated optimized cardinality cuts are far more potent than optimized cardinality cuts. None of the problems in Table 5 could be solved using optimized cardinality cuts only.

The methods used in this study to solve TSP and multidimentional knapsack problems are quite universal. They can be used for any problem that can be formulated as a 0-1 linear integer programming problem. To name a few: set covering, assignment problems with side constraints, scheduling and routing problems, graph coloring, maximum clique are among examples. The approach outlined above may be tailored easily to solve these problems.

Although the computational results presented in this study give strong indication that the methods proposed will expand the limits on sizes of problems solvable by integer programming, there is still need for more experiments.Trying different types of problems and

Table 5
Solution times for randomly generated large problems.

m(Constraints) $\times$ n(variables)	Cardinality cuts *Cpu seconds*	Number of aggregated cuts added	CPLEX *Cpu seconds*	Number of cover cuts added
25×1000	1.45	1	1.23	75
50×1000	765.96	1	2074.46	150
100×1000	10076.20	1	**	**
25×5000	2033.04	1	15221.62	75
50×5000	87.46	1	3438.96	150
100×5000	38448.99	14	**	**
$25 \times 10,000$	4084.70	2	**	**
$50 \times 10,000$	53.19	1	188.10	150
$100 \times 10,000$	21193.43	5 ($K = 300$)	**	**
$25 \times 20,000$	121.05	1	12.47	4
$50 \times 20,000$	32898.56	12 ($K = 500$)	**	**
$100 \times 20,000$	11250.15	4 ($K = 200$)	**	**

the improving the performance of the methods are further research areas.

References

[1] G. Cornuejols, M. Dawande, A class of hard small 0-1 programs, INFORMS Journal on Computing 11 (1999) 205–210.
[2] H. Crowder, E.L. Johnson, M.W. Padberg, Solving large scale 0-1 linear programming problems, Operational Research 31 (1983) 803–834.
[3] G.E. Ferreira, A. Martin, R. Weismantel, Solving multiple knapsack problems by cutting planes, SIAM Journal of Optimization (1996) 6858–6877.
[4] R.E. Gomory, Outline of an algorithm for integer solutions to linear programs, Bulletin of the American Mathematical Society 64 (1958) 275–278.
[5] Z. Gu, G.L. Nemhauser, M.W.P. Savelsbergh, Lifted cover inequalities for 0-1 integer programs: complexity, INFORMS Journal on Computing 11 (1999) 117–123.
[6] E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, Progress in linear programming-based algorithms for integer programming: an exposition, INFORMS Journal on Computing 12 (2000) 2–23.
[7] G.L. Nemhauser, L.A. Wolsey, Integer and Combinatorial Optimization, Wiley, New York, 1988.
[8] OR-Library. Available at (<http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapinfo.html>).
[9] J.E. Beasley, OR-Library: distributing test problems by electronic mail, Journal of the Operational Research Society 41 (11) (1990) 1069–1072.
[10] L.A. Wolsey, Integer Programming, Wiley, New York, 1998.