

Cutting Plane Algorithms for 0-1 Programming Based on Cardinality Cuts

Osman Oguz

Department of Industrial Engineering, Bilkent University, Ankara, Turkey

E-mail: ooguz@bilkent.edu.tr

Abstract: 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.

Keywords: 0-1 integer programming, valid inequalities, cutting planes.

1.Introduction

Crowder et al. [2] demonstrated the important role of cover inequalities used as cutting planes in the branch-and-cut methods for solving integer programming problems. Relatively recent works by Gu et al. [4] and Johnson et al. [5] provide extensive discussions of available strategic choices for using cover inequalities in the branch-and-cut process for 0-1 programming. Nemhauser and Wolsey [6], and Wolsey [7] may be seen for basic expositions of the subject and related issues.

We work on the 0-1 integer programming problem given below to introduce similar inequalities. We have chosen this problem to introduce our approach, because most of the work on cover inequalities are based on this problem. One may see Ferreira et al. [3] for an implementation of cover cuts on the multiple knapsack version of the problem.

$$(\text{IP}) \quad \text{Maximize } z = \sum_{j=1}^n c_j x_j \quad (1)$$

subject to:

$$\sum_{j=1}^n a_{ij} x_j \leq b_i, \quad \text{for } i = 1, \dots, m \quad (2)$$

$$x_j \in \{0,1\}, \quad \text{for } j = 1, \dots, n. \quad (3)$$

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

We do not assume any restrictions on the signs and 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. The preliminary numerical experiments are discussed in Section 3. Conclusions and comments follow in Section 4.

2. The New Cut

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

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

The following inequality is obviously valid:

$$\sum_{j \in S_p} x_j \leq z_0 \quad (5)$$

Also, this inequality is valid for all possible values x_j such that $j \in S_p$ in any solution of the LP relaxation for any objective function. Moreover, the inequality is valid in the form,

$$\sum_{j \in S_p} x_j \leq \lfloor z_0 \rfloor \quad (6)$$

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

$\sum_{j \in S_p} x_j \leq \lfloor z_0 \rfloor$ is not violated by the relaxed solution. Nonetheless, it is the starting point of our proposal for a

new type of cuts.

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: $S_1 = \{j \mid x_j = 1 \text{ in } X_{LP}\}$ and $S_2 = N \setminus S_1$. Then, naming the LP relaxation of the original integer program (IP) as (P1), we define the following linear program named as (P2):

$$(P2) \quad z_1 = \text{maximize} \sum_{j \in S_2} x_j \quad (7)$$

subject to

$$\sum_{j=1}^n a_{ij} x_j \leq b_i, \quad \text{for } i = 1, \dots, m \quad (8)$$

$$\sum_{j \in S_1} x_j = |S_1| \quad (9)$$

$$0 \leq x_j \leq 1, \quad \text{for } j = 1, \dots, n \quad (10)$$

Then the following is true.

Proposition 1: The inequality

$$\sum_{j \in S_1} r x_j + \sum_{j \in S_2} x_j \leq r |S_1| + \lfloor z_1 \rfloor \quad (11)$$

is valid for the solution set of the problem (IP) for sufficiently large values of the parameter r .

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 $\sum_{j \in S_2} x_j \leq \lfloor z_1 \rfloor$. 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 (P2). 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. Assigning a very large value to r 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. ■

Assigning an arbitrarily large value to the parameter r is not desirable since the quality of the cut will be poor, i.e., the cut will remove a very 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 (P2) given below:

$$(P3) \quad z_2 = \text{maximize} \sum_{j \in S_2} x_j \quad (12)$$

subject to

$$\sum_{j=1}^n a_{ij} x_j \leq b_i, \quad \text{for } i = 1, \dots, m \quad (13)$$

$$\sum_{j \in S_1} x_j = |S_1| - 1 \quad (14)$$

$$0 \leq x_j \leq 1, \quad \text{for } j = 1, \dots, n. \quad (15)$$

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 $k \geq 0$. Thus, setting $r = z_2 - \lfloor z_1 \rfloor$ gives a relaxation sufficiently tight for the purpose of cutting deeper into the underlying polyhedron. $z_2 - \lfloor z_1 \rfloor \leq 1$ indicates that any increase in the number of variables with value equal to 1 in S_2 will require at least the same number of variables in S_1 to be equal to 0. We set $r = 1$ arbitrarily in these situations.

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 by Beasley [5]. 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 Eqn.(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 leads 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 Eqn.(11) is a lifted version of that given by Eqn.(6). We take a further step in this direction and obtain what might be called the overlifted version of Eqn.(11), because lifting is done for the purpose of eliminating a certain set of feasible integer solutions from the solution space.

Let's consider the following revised version of **(P2)** defined by equations (7)-(10):

$$\text{(P4)} \quad z_3 = \text{maximize} \quad \sum_{j \in S_2} c_j x_j \quad (16)$$

subject to

$$\sum_{j=1}^n a_{ij} x_j \leq b_i, \quad \text{for } i = 1, \dots, m \quad (17)$$

$$\sum_{j \in S_1} x_j = |S_1| \quad (18)$$

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

The optimal solution of this problem is a feasible solution for **(IP)**. Also, $ZINT_{LB} = z_3 + \sum_{j \in S_1} c_j$ is a lower bound

for the optimal objective function value of **(IP)**. We call the following version of the cardinality cut the optimized cardinality cut:

$$\sum_{j \in S_1} r x_j + \sum_{j \in S_2} x_j \leq r |S_1| \quad (20)$$

Then, we state and prove the following proposition:

Proposition 2: The optimized cardinality cut represented by the inequality in Eqn. (20) makes sure that all solutions of **P4**, 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 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 use a large enough r . ■

Corollary 1: When the inequality of Eqn.(20) is added to the constraint set of **P1**, the LP relaxation of **IP**, with a sufficiently large value of r , 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 **P1**. All other integer feasible solutions remain in the feasible set.

Note that the cut represented by Eqn.(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.

Obviously, we can use this inequality as a cut in the constraint set of problem **P1**, as long as we keep the integer solution corresponding to $ZINT_{LB}$ as an incumbent until it is replaced with a better one, i.e., a solution that gives a higher lower bound. As a proper value of r , we can set $z = z_2$ where z_2 is as defined in Eqn. (12).

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 $\sum_{j \in S_2} x_j \leq \lfloor z_1 \rfloor$ holds, in which case the related inequality can't separate the noninteger solution from the solution set of **P1**, whereas the new version is always able to do the separation. Of course, there is a cost: we have to solve an integer programming problem **P4**, to be able to use the inequality given in Eqn.(20) as a cut. Considering the cardinality of S_2 , the number of variables of the integer programming in question (**P4**) may be quite large, and solving **P4** to optimality may be very time consuming. To reduce the number of variables and the solution time of **P4**, we fix the variables with a value equal to 0 in the LP relaxation of **P1**, just like the variables in S_1 . Fixing is achieved by revising the constraint in Eqn.(18) of **P4** as follows:

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

$$\sum_{j \in S_1} x_j - \sum_{j \in S_0} x_j = |S_1| \quad (21)$$

After solving **P4** with this constraint, the inequality of the optimized cardinality cut given in Eqn.(20) becomes:

$$\sum_{j \in S_1} r x_j - \sum_{j \in S_0} r x_j + \sum_{j \in S_f} x_j \leq r |S_1| \quad (22)$$

The solution time of **P4** is reduced as expected by using Eqn.(21), however there is a drawback: the effectiveness of Eqn.(22) as a cut is much less relative to that of Eqn.(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 **P4**.

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:

$$\text{Cut1:} \quad \sum_{j \in S_1^1} r_1 x_j - \sum_{j \in S_0^1} r_1 x_j + \sum_{j \in S_f^1} x_j \leq r_1 |S_1^1| \quad (23)$$

$$\text{Cut2:} \quad \sum_{j \in S_1^2} r_2 x_j - \sum_{j \in S_0^2} r_2 x_j + \sum_{j \in S_f^2} x_j \leq r_2 |S_1^2| \quad (24)$$

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

$$\text{Cut3:} \quad \sum_{j \in S_1^3} r_3 x_j - \sum_{j \in S_0^3} r_3 x_j + \sum_{j \in S_f^3} x_j \leq r_3 |S_1^3| \quad (25)$$

where:

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

We are now ready to state the following proposition:

Proposition 3: Let $z(1\&2)$ represent the value of the objective function of **P1** solved with Cut1 and Cut2 added as constraints, and $z(3)$ the objective function of the same problem solved with Cut3 only. Then, $z(3) \leq z(1\&2)$ holds.

Proof: Let T_1 be the set of feasible solutions of **P1** with $x_j = 1$ for $j \in S_1^1$, $x_j = 0$ for $j \in S_0^1$, and x_j free 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. Note that a variable fixed at 1 for solving **P4** with Eqn.(21) in generating Cut 1, may be set at 0 in case of Cut 2, assigning this variable to the set of free (not fixed) variables for defining Cut 3. This means Cut3 alone makes a larger subset of feasible solutions of **P1** infeasible than Cut1 and Cut2 combined, and proves the claim. ■

Proposition 3 enables us to throw away Cut1 and Cut2, and solve **P1** 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 **P1**, define Cut1, then add Cut1 to **P1** and re-solve, define Cut2, aggregate Cut1 and Cut2 to obtain Cut3, then replace Cut1 by Cut3, and re-solve **P1**, 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 **P4** to obtain an integer incumbent solution. Once this is done, Cut(K) becomes a permanent constraint of **P1**, we restart the process by solving **P1** 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 **P4** 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 Eqn.(11) is compared with that of cover inequalities on a 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 Eqn.(21) 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 by Chu and Beasley [7]. These problems are inherently difficult, much like but not exactly the same with the market share problems presented by Cornuejols and Dawande [1]. The table below displays the results for the set of first 5 problems from 7 categories of test problems.

The experiments have been carried out on a 2.6GHz, 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). The computational results for 30 problems for each of the categories mknapcb3,4, 5, 6, 7, 8 and 9 are given in Table 1 below. 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. LB is the best lower bound obtained for each problem from solutions of **P2**. GAP is equal to (improvement)/(INITIAL- LB).

We have devised a simple greedy heuristic to find effective cardinality cuts for the purpose of testing the cuts on these 0-1 programming problems as given below:

The Heuristic Algorithm:

Step 1: Solve **P1** and define S_1 and S_f and 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 j^{th} variable in the LP relaxation.

Step 2: Solve **P2**. Let x_j^2 for $j \in S_f$ denote the value of the j^{th} 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 **P1** 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.

	INITIAL	CPLEX			cc				improvement		GAP %	
		improved	time (sec)	Cover Cuts	improved	time (sec)	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
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 1. Results for ORLIB problems with cardinality cuts and cover cuts

A summary of the results in the table above is given below showing the superiority of cardinality cuts over cover inequalities in terms of GAP:

76,67%	percentage superior
29,05%	percentage 5 times superior
17,62%	percentage 10 times superior
9,05%	percentage 20 times superior

Table 2. *Summary results in terms of relative improvements*

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. We have implemented the following simple algorithm:

0. Set $INC = +\infty$ as the value of the incumbent tour.

1. Solve **P1** (2-matching formulation). If the solution is a tour or if the value of the objective function of a **P1** is greater than INC , stop. Otherwise, define S_1 and S_2 .

2. Set $r = N - |S_1|$. Solve **P4**. If the solution is integer, but represents subtours, then add a subtour breaking constraint to **P1** 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 **P1** eliminating the tour and go to step 1. When **P4** is infeasible (no integer solution exists) add the cardinality cut given in Eqn.(21) to **P1** and go to step 1.

The solutions to **P1** and **P4** were obtained using CPLEX version 8.0.1 on the same server.

Problem Name	CPU Seconds
Berlin52.tsp	5.20
Eil51.tsp	50.34
Eil76.tsp	271.13
Eil101.tsp	231.78
Gr96.tsp	505.40
D120.tsp	631.57
Bier127.tsp	2369.62
Ch130.tsp	1866.62
Ch150.tsp	5252.53

Table 3. *Solution times for some TSPLIB problems*

Our attempts to solve larger problems were not successful due to excessive solution times for the subproblem **P4**.

4.3 Experiments with the Multidimensional Knapsack Problems

The cutting plane algorithm used to solve the multidimensional knapsack problems in this subsection uses the aggregated cardinality cut given in Eqn.(25). An outline of the algorithm is as follows:

0. Set $INC = -\infty$ as the value of the incumbent solution, $k=0$, and assign a value to the aggregation frequency $1 \leq K \leq 100$.

1. Solve **P1**(the LP relaxation). If the solution is integer, stop. If the value of the objective function of **P1** is smaller than INC , go to Step 4. Otherwise, define S_1, S_2 and S_f .

2. Solve the problem:

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

to determine the value of 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 r and current partition. Add this cut to P_1 , set $k=k+1$ and go to step 1. If $0 < k < K$, go to the next step. Otherwise go to step 4.

3. Carry out an aggregation procedure as described by Eqn.s(23,24,25) 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 k^{th} partition. Determine r with the k^{th} partition and replace the last added cut to P_1 by the new cut obtained using the k^{th} partition and go to step 1.

4. Carry out an aggregation procedure combining $K-1^{\text{st}}$ partition with the K^{th} (the result is still called the K^{th} partition), and solve P_4 . If the solution is better than the incumbent, record it as the new incumbent. When P_4 is infeasible (no integer solution exists), replace the $K-1^{\text{st}}$ cut by the K^{th} in P_1 , set $k=0$ and go to step 1.

Implementation of this algorithm on the Chu and Beasley (1998) problems gave the results listed in Table 4 below. We have used $K=30,50,50$ for problems with 100, 250 and 500 variables correspondingly.

Problem Name	Cardinality Cuts	CPLEX
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

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

The data in Table 4. displays the time performance of the algorithm described at the beginning of this subsection with that of CPLEX (*Interactive Optimizer 8.1.0*, Copyright (c) ILOG 1997-2002) on some of the OR-Library problems of J. E. Beasley given at:

(<http://people.brunel.ac.uk/~mastjjb/jeb/info.html>).

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 (*mknpcb3_30*, *mknpcb5_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. Multidimensional 0-1 knapsack problems were generated in the range of 1,000-20,000 variables and 25-100 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 beginning of this subsection together with time performance of CPLEX on ten 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.

M x N	Cardinality Cuts <i>Cpu seconds</i>	CPLEX <i>Cpu seconds</i>
25 x 1,000	19.71	124.15
50 x 1,000	1365.30	4420.92
100 x 1,000	2677.89	10911.12
25 x 5,000	925.29	4607.0
50 x 5,000	3503.63	*****
100 x 5,000	3857.86	*****
25 x 10,000	7662.47	*****
50 x 10,000	8733.66	*****
100 x 10,000	15264.24	*****
25 x 20,000	551.72	*****

Table 5. *Solution times for randomly generated large problems.*

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 3 provide evidence that these cuts may be more effective than cover cuts: the main force 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 not optimal. These cuts were used to

solve a small set of TSP test problems from the literature. Very promising computational results are displayed in Table 3.

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. The last problem with 20,000 variables in Table 5, for example, was solved by solving a 0-1 integer program with 242 variables and adding a single cut to the original problem. This may be an extreme case, although the number of cuts added never exceeded ten in any of the test problems in Table 5.

Main feature of the approach is the use of the information obtained from the solution of the subproblem **P4**. The solution provides a potential incumbents 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.

The methods used in this study to solve TSP and multidimensional 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 the improving the performance of the methods are further research areas.

References:

1. Cornuejols G., M. Dawande 1999, A Class of Hard Small 0-1 Programs, *INFORMS Journal on Computing* 11, 205-210.
2. Crowder, H., E. L. Johnson, M. W. Padberg 1983. Solving large scale 0-1 linear programming problems. *Oper. Res.* 31, 803-834.
3. Ferreira G. E., Martin A., R. Weismantel 1996. Solving multipl knapsack problems by cutting planes, *SIAM Journal of Optimization* 6 **858-877**.
4. Gu, Z., G. L. Nemhauser, M. W. P. Savelsbergh 1999. Lifted Cover Inequalities for 0-1 Integer Programs: Complexity, *INFORMS Journal on Computing* 11, 117-123.
5. Johnson, E. L., G. L. Nemhauser, M. W. P. Savelsbergh 2000. Progress in Linear Programming-Based Algorithms for Integer Programming: An Exposition. *INFORMS Journal on Computing* 12, 2-23.
6. Nemhauser, G. L., L. A. Wolsey 1988. Integer and Combinatorial Optimization. Wiley, New York.
7. OR-Library. Available at (<http://people.brunel.ac.uk/~mastjib/jeb/orlib/mknapiinfo.html>). OR-Library was originally described in J.E.Beasley, "OR-Library: distributing test problems by electronic mail", *Journal of the Operational Research Society* 41(11) (1990) pp1069-1072
8. Wolsey, L. A. 1998. Integer Programming. Wiley, New York.

