SEARCH AND CUT: NEW CLASS OF CUTTING PLANES FOR 0-1 PROGRAMMING

Osman Oğuz
Department of Industrial Engineering
Bilkent University
Ankara, Turkey

Abstract The basic principle of cutting plane techniques is to chop away the portions of the solution space of the linear programming relaxation of an integer program which contain no integer solutions. This is true for both Gomory's cutting planes, and other more recent cuts based on valid inequalities. Obtaining a partial or full description of the convex hull of the set of integer solutions is the underlying motivation for these approaches. The cuts proposed in this study break away from this motivation, and allow for the chopped away portions of the solution space to have both feasible and infeasible integer points.

Subject Classification: Integer Programming
Key Words: 0-1 Integer Programming, Cutting Planes, Branch and Cut.

1 Introduction

A well known fact is that, it is theoretically possible to solve any integer programming problem using the fractional cuts developed by Gomory (1958). Also a well known fact is that, the performances of the techniques based only on the use of Gomory's cuts, are hopelessly short of the practical requirements to solve problems encountered in real life. Other more potent cutting planes, like cover inequalities (a special class of valid inequalities for 0-1 programming problems), also have their shortcomings. Cut generation may sometimes be as difficult as solving the integer program itself. However, they were the basis for the development of the most successeful technique for solving integer programs, namely, branch-and-cut.

These techniques and the related literature have been treated in detail in the books by Nemhauser and Wolsey (1988), and Wolsey (1998). To study the development of the theory in more detail and issues related to its implementation, one can see Balas (1975), Balas and Zemel (1978), Hammer, Johnson and Peled (1975), Wolsey (1976), Zemel (1978), Nemhauser and Vance (1994). The first report of the implementation of using valid inequalities together with branching (branch-and-cut) was given by Crowder, Johnson and Padberg (1983). Padberg and Rinaldi (1993) solved a large traveling salesman problem using branch-and-cut. Gu, Nemhauser and Savelsberg (1998) provide a discussion of best strategies for the use of cover inequalities in branch-and-cut, and extensive computational results. Balas, Ceria and Cornuejols (1996) raise the point that the cover inequalities may not be sufficiently strong because they are derived from just one constraint, and treat the use of Gomory's fractional cuts in branch and cut. Cornuejols and Dawande (1999) report the poor performance of branch-and-cut on a specific problem class.

The idea of searching (possibly by total enumeration) a certain portion of the solution space of an integer programming problem to generate cutting planes is not new. Balas (1972), Burdet (1972), and Glover (1972) used this idea in the early seventies to generate Gomory style cuts. The intersection cuts proposed by Balas (1972) were recently used in the development of a new heuristic by Balas et al. (2001).

Here, in this study we propose a mechanism which generates cuts for 0-1 integer programming problems considering the solution space of the 0-1 programming problem in the neighborhood of a candidate solution. We enumerate the neighborhood around the candidate solution using k-dimensional search, where k is expected to be an integer much smaller than n, the number of 0-1 variables in the problem. The candidate solution is an integer point in the neigborhood of the fractional solution to a linear programming

relaxation of the problem. Unlike Balas, Burdet, and Glover, this integer candidate solution is the basis of the neigborhood search , not the fractional solution at hand. Thus, we can say that the approach taken in this study generates cuts more akin to cover inequalities (both methods provide inequalities with 0-1 coefficients only) rather than fractional cuts. This choice enables us to use a simpler and more direct enumerative search.

We give the details of the method in the next section, furnishing the discussion with a formal algorithm. In Section 3, we discuss a further refinement of the method to enhance its effectiveness. Two illustrative numerical examples are given in Section 4. The results of some computational experimentation with test problems both from literature and from our own random problem generator are reported in Section 5. Section 6 is devoted to the discussion of issues related to implementation of the method and conclusions.

2 The Description of the Search Based Cuts

We consider the following 0-1 programming problem:

$$\text{Maximize} \sum_{j=1}^{n} c_j x_j \tag{1}$$

Subject to:

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i, \ \forall \ i = 1, .., m \tag{2}$$

$$x_j \ \in \ \{0, 1\} \ \forall \ j = 1, .., n \tag{3}$$

Assume that all problem data are integers. Let $X^* = (x_1^*, x_2^*, .., x_n^*)$ denote a solution to the linear programming (LP) relaxation of the problem. That is, we replace the constraint set imposed by $x_j \ \in \ \{0, 1\} \ \forall \ j = 1, .., n$ by the set $0 \leq x_j \leq 1, \ \forall j = 1, .., n$, and solve the problem by a linear programming solver. Then, an n dimensional 0-1 vector $X_{int} = (x_1', x_2', .., x_n')$ is generated using X^* by the following definition:

$$x_j' \ = \begin{cases} 1 & \text{if } x_j^* \ \geq \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}$$

The vector X_{int} is our candidate solution.
It may be feasible if $\sum_{j=1}^{n} a_{ij} x_{ij}' \leq b_i, \ \forall \ i = 1, .., n$ hold. It may even be optimal with a small likelihood. It is most likely that it represents an infeasible integer solution. Denoting $S_1 = \{j | x_j' = 1\}$, and $S_2 = \{j | x_j' = 0\}$, we have:

$$\sum_{j \in S_1} x_j' \ + \ \sum_{j \in S_2} (1 - x_j') \ = \ n \tag{4}$$

Now, if X_{int} represents an infeasible solution to (1)-(3), and $X = (x_1, x_2, .., x_n)$ represents a feasible solution for the same set, then:

$$\sum_{j \in S_1} x_j \; + \; \sum_{j \in S_2} (1 - x_j) \; \leq \; n - 1 \tag{5}$$

must hold, that is, at least one x_j must be different than its counterpart x'_j for $j \in S_1 \cup S_2$.

Consider now, carrying out a one dimensional search on the vector X_{int} given by the procedure described above. That is, if $x'_1 = 1$, change its value to zero, keeping all other components of the vector at their current values, test the resulting vector for feasibility. If $x'_1 = 0$, change its value to 1, and do the same test. If the test is repeated n times for all components x'_j, $\forall j = 1, .., n$ and if no feasible solution is found, then we can reduce the right hand side constant of the inequality (5) to $n - 2$, and the inequality will still be valid since it removes no feasible 0-1 vector from the solution space. Obviously, if we carry out two dimensional search, that is, change the candidate vector in all possible pairs of directions and find no feasible solution, we can decrease the right hand side of the inequality from $n - 2$ to $n - 3$. Generalizing, if t dimensional searches have been done for $t = 1, .., k$, resulting in the discovery of no feasible solutions, then the inequality given in (5) is a valid inequality for the 0-1 programming problem with the right hand side value of $n - k - 1$.

Going one step further, one may ask what happens if one or more feasible solutions are encountered during the search process? The answer is simple: just keep a record of the best solution encountered as an incumbent, use the inequality with the right hand side constant determined by the depth k of the search, as a cut. We are now at a position to give a finite algorithm which may be quite inefficient, but can solve the problem given in (1)-(3). We need the following definition to facilitate the ensuing exposition.

Definition: The difference $\delta = n \; - \; \sum_{j=1}^{n} \max[x_j^*, \; (1 - x_j^*)]$ is called *the integrality gap* associated with the vector X^*.

The smallest possible value for the integrality gap is zero, implying X^* is integer. Its largest possible value is $\frac{n}{2}$ which happens when all x_j^* for $j = 1, .., n$ are equal to $1/2$. Also, for the valid inequality resulting from a search with depth k to be a cut, we must have:

$$k \geq \lfloor \delta \rfloor \tag{6}$$

We choose to set $k > \lfloor \delta \rfloor$ to make the resulting cuts deeper in the algorithm given below.

The Search & Cut Algorithm

Step 0. Set the value of the incumbent solution value $\bar{z}$ to $-\infty$.

Step 1. Solve the linear programming relaxation of the problem given in (1)-(3) plus any

cuts generated so far to get X^*. Stop, if either the problem is infeasible, or the value of the objective fuction for X^* is less than $\bar{z}+1$, declaring the current incumbent solution as optimal.
Step 2. Compute δ, and set the value of $k > \lfloor \delta \rfloor$.
Step 3. Perform t-dimensional search on the vector X_{int}, generated from X^*, for $t = 1, 2, .., k$. Each time a feasible solution is found in the process, compare its value with $\bar{z}$. If it is higher than $\bar{z}$, set $\bar{z}$ equal to this new value, and replace the incumbent solution vector by the feasible solution found.
Step 4. Append a new cut and go to Step 1.

The finiteness of the algorithm is guaranteed by the fact that there are only 2^n possible solution vectors. Some vectors may recur. The recurrence is possible due to the fact that the noninteger vectors $X^1 = (1, 1, 1, 0.49, 0.49, 0.49)$ and $X^2 = (1, 1, 0.5, 0, 0, 0)$ will both lead to the integer candidate solution $X_{int} = (1, 1, 1, 0, 0, 0)$. If we use a search depth of k for the first ocurrence, then we have to use at least $k+1$ for the second time. Thus the maximum number of recurrences per solution vector cannot exceed $n-1$, guaranteeing the finiteness of the algorithm.

3 Partitioning the Problem and the Integrality Gap to Get a Cut

A major drawback of the algorithm is its dependence on enumerative search to generate a cut. The basic parameter determining the amount of work needed for this process is δ; the integrality gap. It is clear that the efficiency of the algorithm will rapidly deteriorate as the value of δ gets larger. For a problem with a couple of hundred variables, using a search depth of $k = 10$ may require a prohibitive amount of computational time to get a cut. We propose the following method to bring down this computational load within acceptable limits.

Suppose that we have a subset T of the the components of the vector X^* such that:

$$T \subset N = \{1, 2, .., n\} \text{ , and } |T| - \sum_{j \in T} \text{Max}[x_j^*, (1 - x_j^*)] < 1 \tag{7}$$

Then, if

$$\text{Max } t = \sum_{j \in N \setminus T} c_j x_j \tag{8}$$

Subject to:

$$\sum_{j \in N \setminus T} a_{ij} x_j \ \leq \ b_i - \sum_{j \in T} a_{ij} x'_j \ \forall i = 1, .., m \tag{9}$$

and

$$x_j = 0 \text{ or } 1 \ \text{ for } j \in N \setminus T \tag{10}$$

has no solution, the following inequality is valid for the solution set of our problem:

$$\sum_{j \in T \bigcap S_1} x_j + \sum_{j \in T \bigcap S_2} (1 - x_j) \leq |T| - 1 \tag{11}$$

And, by the assumption made for Eqn. (7), this valid inequality separates X^* from the set of feasible solutions to the LP relaxation of the problem. Again, it is possible to improve the quality of this cut by carrying out one dimensional search on the components of X_{int} in the set T. If we can show that the problem given in (8)-(10) has no solution for all $|T|$ problems obtained by switching each of the variables from 0 to 1, or vice versa one at a time, then we can reduce the right hand side of the inequality from $|T| - 1$ to $|T| - 2$ to obtain a deeper cut.

As before, if we come across with a feasible solution, i.e., the solution to (8)-(10) is feasible, then we compare the value of $z = t + \sum_{j \in T} c_j x'_j$ with the value of the incumbent solution and store this value and the corressponding solution as the new incumbent if it is better. When the one dimensional search is over, we generate and use the cut in the same fashion as described for the all tested solutions infeasible case.

We have already pointed out that it is conceivable to have a solution with $\delta = \frac{n}{2}$, in which case, the cost of getting a cut by the search mechanism proposed in this study will be extremely prohibitive for any problem with moderate size. This happens when $x^*_j = 1/2$ for $j = 1, .., n$.

We will propose an approache to get over this difficulty, i.e., what to do with instances of large δ. We give the algorithm to determine the set T, before doing so.

Algorithm to obtain T:

Step 1. Order the components x^*_j of the vector X^* with increasing $\text{Max}[x^*_j, (1 - x^*_j)]$ values so that:

$|x^*_{j(i)} - \frac{1}{2}| \leq |x^*_{j(i+1)} - \frac{1}{2}|$. Set $Q = \{\emptyset\}$.

The subscript $j(i)$ means that the j'th component is in the i'th position in the ordering.

Step 2. Set $k = 1$.

Step 3. Check if $n \ - \sum_{i=k+1}^{n} \max[x^*_{j(i)}, \ (1 - x^*_{j(i)})] \ \geq 1$ is true. If true, set $Q = Q \bigcup \{j(k)\}$, $k = k + 1$ and $n = n - 1$ and repeat Step 3. Otherwise go to Step 4.

Step 4. Set $T = N \setminus Q$ and stop.

We now go one more step further in this direction by repeatedly partitioning subproblems in order to solve smaller integer programming problems to get cuts for the original problem. That is, use the partitioning scheme outlined above recursively, emulating the depth-first version of branch and bound algorithm to some extent, as explained below:

Let us reconsider the problem depicted by Eqns. (8)-(10). Note that the depiction is based on the set T. Let's call the problem $\mathbf{P}(T_0)$ by subscripting T as the initial set. As was made clear earlier, this problem may still have a pretty large number of variables, thus being only slightly less challenging than the original problem. Then, instead of solving it directly as we had suggested above, we may attempt to solve this problem with the aid of the new cuts we are proposing. That is, we solve the LP relaxation of this problem to define a candidate solution vector and the associated integrality gap. Then, applying the above algorithm, define T_1 and $\mathbf{P}(T_1)$. Note that $T_1 \subset N \setminus T_0$, and $|T_1| - \sum_{j \in T_1} \text{Max}[x_j^*, (1 - x_j^*)] < 1$. The subproblem $\mathbf{P}(T_1)$ is defined as:

$$\text{Max } t_1 = \sum_{j \in N \setminus T_0 \cup T_1} c_j x_j \tag{12}$$

Subject to:

$$\sum_{j \in N \setminus T_0 \cup T_1} a_{ij} x_j \ \leq \ b_i - \sum_{j \in T_0 \cup T_1} a_{ij} x_j' \ \forall i = 1, .., m \tag{13}$$

and

$$x_j = 0 \text{ or } 1 \ \text{ for } j \in N \setminus T_0 \cup T_1 \tag{14}$$

When $\mathbf{P}(T_1)$ is resolved, i.e., an optimal solution to it is found or it is declared infeasible, then, as explained earlier, a cut is added for $\mathbf{P}(T_0)$. If, on the other hand we think that the set $N \setminus T_0 \cup T_1$ is still too large, we can repartition it by defining T_2 and $\mathbf{P}(T_2)$ whose solution will lead to a cut for $\mathbf{P}(T_1)$. Generalizing, suppose that we have used partitioning recursively to reach $\mathbf{P}(T_i)$, which has only a few variables, thus very easy to solve. After solving $\mathbf{P}(T_i)$, we move back and add a new cut to $\mathbf{P}(T_{i-1})$, and solve its LP relaxation. Note that, all books pertaining to $\mathbf{P}(T_i)$ are discarded, and the set T_i is undefined at this point. We redefine $\mathbf{P}(T_i)$ using the new LP relaxation, and solve it to generate a new cut. This process continues until $\mathbf{P}(T_{i-1})$ is solved, i.e., its LP relaxation is either integer, or has an optimal objective function value less than or equal to that of the incumbent solution, or infeasible. In all three cases, the result is a new cut for $\mathbf{P}(T_{i-2})$. When sufficient cuts are added to solve $\mathbf{P}(T_{i-2})$, we generate a cut for $\mathbf{P}(T_{i-3})$. The process moves back and forth recursively until we reach a solution for $\mathbf{P}(T_0)$, in which case, a cut for the original problem is generated. At this point, the whole process begins with

the computation of a new X_{LP} and X_{int}, and a fresh round of partitioning starting with redefining T_0.

Note that the procedure outlined is very similar to what is done in a depth first branch and bound algorithm. The variables in the set $R = \bigcup_{k=0}^{i} T_k$ are fixed to either 0 or 1, corresponding to a node of the branch and bound tree, and the node is fathomed by solving $\mathbf{P}(T_k)$. In the backtracking stage, however there is no definite path to follow. The new node to be fathomed is determined by solving the LP relexation of $\mathbf{P}(T_{k-1})$ with a new cut. Also, the information pertaining to $\mathbf{P}(T_k)$ is discarded as soon as a new cut is generated. That is, there is no need to store information related to a set of live nodes because the cuts we add determine which node to be fathomed next. Rather than giving a formal description of the procedure just outlined, we illustrate its application with a small numerical example in the next section.

4 Numerical Examples

Example 1

The example problem is taken from Nemhauser and Wolsey. It is solved by one stage search and cut procedure, i.e., the recursive partitioning is not used:

Maximize z = $77x_1 + 6x_2 + 3x_3 + 6x_4 + 33x_5 + 13x_6 + 110x_7 + 21x_8 + 47x_9$

Subject To:

$774x_1 + 76x_2 + 22x_3 + 42x_4 + 21x_5 + 760x_6 + 818x_7 + 62x_8 + 785x_9 <= 1500$

$67x_1 + 27x_2 + 794x_3 + 53x_4 + 234x_5 + 32x_6 + 797x_7 + 97x_8 + 435x_9 <= 1500$

$x_j = 0$ or 1 for $j = 1, .., 9$

The solution of the LP relaxation of the problem is:

$x_4 = x_5 = x_7 = x_8 = 1,\ x_1 = 0.71,\ x_3 = 0.35,\ x_2 = x_6 = x_9 = 0$ with $z = 225.7$

The integrality gap δ associated with this solution is equal to 0.94. This means $k = 0$ provides us with an effective cut without the need for any search. However, for the sake of obtaining deeper cuts, we choose the default value of $k = 1$.

The candidate solution is: $X_{int} = (1, 0, 0, 1, 1, 0, 1, 1, 0)$ which turns out to be infeasible. One dimensional search on this vector also fails to give any feasible solution. Thus, we get the following valid inequality:

$x_1 + (1 - x_2) + (1 - x_3) + x_4 + x_5 + (1 - x_6) + x_7 + x_8 + (1 - x_9) \leq 7$

which simplifies to: $x_1 - x_2 - x_3 + x_4 + x_5 - x_6 + x_7 + x_8 - x_9 \leq 3$

We add this cut to the problem as the third constraint and resolve the LP relaxation to get:

$x_2 = x_5 = x_7 = x_8 = 1, x_1 = 0.63,\ x_3 = 0.33,\ x_4 = 0.71,\ x_6 = x_9 = 0$ with $z = 223.6$
The integrality gap δ=0.99. Again, $k = 0$ is sufficient, but we use the default value $k = 1$ on the candidate solution $X_{int} = (1, 1, 0, 1, 1, 0, 1, 1, 0)$, which is again infeasible. A one dimensional search on this vector yields $X = (0, 1, 0, 1, 1, 0, 1, 1, 0)$ as the best integer solution (incumbent) with the objective function value $z = 176.0$. All of the cuts used until the objective function value of the LP relaxation falls under this value, are listed below:

cut1:$x_1 - x_2 - x_3 + x_4 + x_5 - x_6 + x_7 + x_8 - x_9 \leq 3$
cut2:$x_1 + x_2 - x_3 + x_4 + x_5 - x_6 + x_7 + x_8 - x_9 \leq 4$
cut3:$x_1 - x_2 - x_3 + x_4 + x_5 - x_6 + x_7 + x_8 - x_9 \leq 2$
cut4:$x_1 - x_3 + x_5 - x_6 + x_7 - x_9 \leq 2$
cut5:$-x_3 + x_5 - x_6 + x_7 + x_8 - x_9 \leq 2$
cut6:$x_1 + x_3 + x_5 - x_6 + x_7 + x_8 - x_9 \leq 3$
cut7:$-x_3 + x_4 + x_5 - x_6 + x_7 - x_9 \leq 2$
cut8:$-x_2 - x_3 + x_5 - x_6 + x_7 - x_9 \leq 0$
cut9:$x_2 - x_3 + x_5 - x_6 + x_7 + x_8 \leq 2$
cut10:$x_1 + x_2 + x_3 + x_4 + x_5 - x_6 + x_7 - x_8 - x_9 \leq 4$
cut11:$x_1 + x_2 + x_3 - x_6 + x_7 - x_8 - x_9 \leq 2$
cut12:$x_1 + x_2 + x_4 - x_5 - x_6 + x_7 + x_8 - x_9 \leq 3$
cut13:$x_1 + x_4 - x_6 + x_7 - x_9 \leq 1$
cut14:$x_1 - x_4 + x_5 - x_6 \leq 0$
cut15:$-x_1 - x_2 - x_5 - x_6 + x_7 + x_8 \leq 0$
cut16:$-x_1 + x_7 + x_8 - x_9 \leq 0$

The partitioning procedure was implemented for all the cuts above with less than 9 variables. One dimensional search has been used in all cases. First feasible incumbent solution turned out to be the optimal solution.

Example 2

The use of recursive partitioning is demonstrated in this example.
Maximize $z = 18x_1 + 22x_2 + 14x_3 + 36x_4 + 17x_5 + 14x_6 + 8x_7 + 24x_8 + 14x_9 + 7x_{10}$
Subject To:
$x_2 + x_3 \leq 1$
$x_4 + x_5 + x_6 \leq 1$
$x_2 + x_4 + x_8 + x_9 \leq 1$
$x_3 + x_4 + x_8 + x_{10} \leq 1$
$x_1 + x_2 + x_7 + x_9 \leq 1$
$x_1 + x_2 + x_4 + x_6 + x_8 \leq 1$

$x_j = 0$ or 1 for $j = 1, .., 10$

The LP relaxation of the problem has the solution:

$X_{LP} = (.33 \quad .33 \quad .67 \quad 0 \quad 1 \quad 0 \quad 0 \quad .33 \quad .33 \quad 0)$, with $Z_{LP} = 52.33$. The candidate integer solution based on X_{LP} is $X_{int} = (0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0)$, which is feasible and becomes the first incumbent solution with $Z_I = 31$. The integrality gap associated with X_{LP} is $\delta = 1.6$. We partition the problem with $T_0 = \{4 \quad 5 \quad 6 \quad 7 \quad 10\}$, fixing $x_4 = 0,\ x_5 = 1,\ x_6 = 0,\ x_7 = 0,\ x_{10} = 0$, and defining $\mathbf{P}(T_0)$ as:

Maximize $z(T_0) = 18x_1 + 22x_2 + 14x_3 + 24x_8 + 14x_9$

Subject To:

$x_2 + x_3 \leq 1$

$x_2 + x_8 + x_9 \leq 1$

$x_3 + x_8 \leq 1$

$x_1 + x_2 + x_9 \leq 1$

$x_1 + x_2 + x_8 \leq 1$

$x_j = 0$ or 1 for $j \in \{1, 2, 3, 8, 9\}$

Note that, we chose T_0 so that, $\sum_{j \in T_0} \text{Max}[x_j^*, (1 - x_j^*)] \leq |T_0| - 1$ becomes a valid cut when $\mathbf{P}(T_0)$ is solved. The same logic is used in choosing the subsequent partitions. The LP relaxation of this problem is denoted by X_{LP}^0 with $x_1 = .33,\ x_2 = .33,\ x_3 = .67,\ x_8 = .33,\ x_9 = .33$. The candidate solution X_{int}^0 for this subproblem is $x_1 = 0,\ x_2 = 0,\ x_3 = 1,\ x_8 = 0,\ x_9 = 0$. Repartitioning $\mathbf{P}(T_0)$ with $T_1 = \{1\}$, fixing $x_1 = 0$, and defining $\mathbf{P}(T_1)$ as:

Maximize $z(T_1) = 22x_2 + 14x_3 + 24x_8 + 14x_9$

Subject To:

$x_2 + x_3 \leq 1$

$x_2 + x_8 + x_9 \leq 1$

$x_3 + x_8 \leq 1$

$x_2 + x_9 \leq 1$

$x_2 + x_8 \leq 1$

$x_j = 0$ or 1 for $j \in \{2, 3, 8, 9\}$

The corresponding LP relaxation X_{LP}^1 is given as $x_2 = .5,\ x_3 = .5,\ x_8 = .5,\ x_9 = 0$, and the accompanying candidate solution X_{int}^1 for this subproblem is $x_2 = 1,\ x_3 = 1,\ x_8 = 1,\ x_9 = 0$. Next, we repartition $\mathbf{P}(T_1)$ with $T_2 = \{9\}$, x_9 fixed at 0 to get $\mathbf{P}(T_2)$ as:

Maximize $z(T_2) = 22x_2 + 14x_3 + 24x_8$

Subject To:

$x_2 + x_3 \leq 1$

$x_2 + x_8 \leq 1$

$x_3 + x_8 \leq 1$
$x_j = 0$ or 1 for $j \in \{2, 3, 8, \}$
The LP relaxation to this problem has, of course, $x_2 = .5,\ x_3 = .5,\ x_8 = .5$ as the optimal solution, which leads us to repartition of $\mathbf{P}(T_2$ with $T_3 = \{2\}$, fixing $x_2 = 1$. The resultant problem is $\mathbf{P}(T_3$:
Maximize $z(T_3) = 14x_3 + 24x_8$
Subject To:
$x_3 \leq 0$
$x_8 \leq 0$
$x_j = 0$ or 1 for $j \in \{3, 8, \}$
which has the unique integer solution $x_3 = x_8 = 0$. This brings us to the end of forward recursions, i.e., the node corresponding to fixing the variables in the set $T_0 \cup T_1 \cup T_2 \cup T_3$ has now been fathomed with the optimal integer solution of $\mathbf{P}(T_3$, which is $x_3 = x_8 = 0$ with the objective function value of $z(T_3) = 0$. Now, all ten variables are assigned values which constitutes the feasible integer solution $X = (0\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0)$ with $z = 39$. This is a better solution, so it becomes the current incumbent.

We now start the backtracking by defining a cut for $\mathbf{P}(T_2)$ using the set T_3. Since $|T_3| = 1$, there is only one possible cut given by $x_2 \leq 0$. Appending this cut to $\mathbf{P}(T_2)$, and solving its LP relaxation gives the integer solution $x_2 = 0,\ x_3 = 0,\ x_8 = 1$, which fathoms the node corresponding to the set $T_0 \cup T_1 \cup T_2$. The optimal integer solution at this node is $X = (0\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0)$ with $z = 41$. This solution is now the incumbent. Having been done with $\mathbf{P}(T_2)$, we generate the cut $x_9 \geq 1$ from T_2, append it to $\mathbf{P}(T_1)$ to obtain $x_2 = x_8 = 0$, $x_3 = 1$ from the LP relaxation; as another optimal integer solution which fathoms the node $T_0 \cup T_1$. The overall integer solution is $X = (0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 0)$ with $z = 45$, becoming the new incumbent.

We then add the constraint $x_1 \geq 1$ to $\mathbf{P}(T_0)$, and solve its LP relaxation, obtaining $x_1 = x_3 = 1, x_2 = x_8 = x_9 = 0$ as its solution. This fathoms the node corresponding to the set T_0, with the overall solution $X = (1\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0)$ with $z = 49$, which becomes the new incumbent.

Next, we generate the cut $-x_4 + x_5 - x_6 - x_7 - x_{10} \leq 0$ from T_0, append it to the original problem, and solve the LP relaxation. We see that the value of the objective function for the noninteger solution is 49.8, which leaves no room for improving the current integer incumbent solution. We stop and declare the current incumbent as an optimal solution of the problem. Note that, if the relaxed solution had an objective function value greater than or equal to 50, a new round of forward recursions would be necessary.

5 Computational Experiments

We report now the results of experimenting with the some of the ideas described in the preceding sections. Unfortunately, recursive partitioning is not one of the tested ideas. We first chose a set of test problems both from literature, and from our own random problem generator. Our major goal in running these experiments was to get a preliminary indication about the performance of the cuts we have proposed. The necessary programming was done in C programming language to provide the interface with CPLEX Linear Optimizer 5.0 (1997) to solve LP relaxations of the 0-1 programs, and to solve the small scale 0-1 programs that arise in the course of the search phase of the method. Our proficiency in C language is quite modest, and the code we have prepared can certainly be improved by an experienced programmer.

The program executes basically the following operations. It reads the original data and sends it to the LP solver to get the solution to LP relaxation. Then it computes the integrality gap and generates the candidate solution. If the gap is greater or equal to 3, it carries out only a one stage partitioning procedure. Otherwise it performs direct enumeration with $k = 2$. The same search depth value is used for enumeration on the set T_0 when one stage partitioning is carried out. A cut is defined after the enumeration process in both cases, and the cut is appended to the constraint set and the LP solver is called upon to provide a new solution.

The test set includes 28 problems. The first 4 are from MIPLIB by Bixby et al. (1998). We chose these problems because they were the only small sized pure 0-1 programming problems in the set. The next 14 are taken from *SAC-94 Suite of 0-1 Multi-Knapsack Problem Instances* which contain problems from Freville and Plateau (1990), Petersen (1967), Shi (1979), and Weingartner and Ness (1967). The third subset of problems are randomly generated multi-knapsack and set covering problems, with the exception of *book.lp* , which is taken from Garfinkel and Nemhauser (1972). The coefficients for the objective function and the costraints are uniformly distributed between 0 and 1000. The right hand side values are uniformly distributed between $100n$ and $100n + 1000$. The set covering problems have all 0-1 cofficients with equal probability of being 0 or 1. The right hand side values are all equal to one. The computer runs were made on a SUN Enterprise 4000 (248 Mhz).

The summary of the results are exibited in Table 1. The columns headed Vars, Rows, Z_{lp} and Z_{IP} correspond to number of variables (n), number of constraints (m), optimal value of the objective function for the LP relaxation, and the optimal value of the integer solution, respectively. The column Time stands for the *cpu* times, and Num. cuts added

column shows how many cuts were generated before the algorithm stops. Other two columns are self explanatory.

	Name	Vars	Rows	Zlp	Zip	Time (sec.)	# cuts added	Opt. Soln. Found?	Optimality Verified?
	bm23	27	20	20.3	34	3444	67	YES	YES
Set 1	p0033	32	15	2828.33	3089	-	109	YES	NO
	p0040	40	23	50794.5	50852	58	2	YES	YES
	pipex	48	25	773.75	788.26	-	92	YES	NO
	weign1	28	2	142019	141278	171	36	YES	YES
	weign2	28	2	131637.5	130883	68	13	YES	YES
	weign3	28	2	99647.08	95677	480	35	YES	YES
	weign4	28	2	122505.3	119337	1653	65	YES	YES
	weign5	28	2	100433.2	98796	196	26	YES	YES
	weign6	28	2	131335	130623	97	19	YES	YES
Set 2	weign7	105	2	1095721	1095445	12845	40	YES	YES
	weish01	30	5	4632.26	4554	2080	40	YES	YES
	weish02	30	5	4592.69	4536	575	45	YES	YES
	weish28	90	5	9514.18	9492	1710	17	YES	YES
	weish29	90	5	9429.03	9410	815	9	YES	YES
	weish30	90	5	11194.69	11191	23	6	YES	YES
	hp1	28	4	3472.34	3418	21915	258	YES	YES
	pet2	10	10	93004.56	87061	143	23	YES	YES
	mk10-10	10	10	1130.96	834	3	3	YES	YES
	mk20-10	20	10	2580.3	2406	74	10	YES	YES
	mk30-10	30	10	5585.14	5225	447	25	YES	YES
	mk40-10	40	10	7702.76	7520	137	7	YES	YES
Set 3	mk50-50	50	50	7056.91	6549	10840	45	YES	YES
	set25-25.lp	25	25	1.974	3	35	2	YES	YES
	set50-50	50	50	2.015	3	18000	89	YES	YES
	set100-10.lp	100	10	1.3636	2	692	5	YES	YES
	book.lp	15	35	5	9	158	15	YES	YES

Table 1 Test Problems and Results obtained by the algorithm

We did not collect statistics for the integrality gap value δ. We only know that maximum number of variables for the 0-1 programming problems solved by using CPLEX during the search phase was 13. These limited computational experiments provide some positive indications about the potential usefulness of the new cuts. All test problems except two *(p0033, pipex)* were solved by using the proposed cuts only. The computer runs for these two cases were terminated prematurely due to excessive computing time. No branching was carried out except for the solution of small scale problems involved in the search phase.

6 Conclusions and Remarks

The methods proposed in this paper may be new additions to our arsenal in dealing with integer programming problems. The new cuts may turn out to be a viable alternative when other possibilities like Gomory cut or cover inequalities seem to be poor choices. The flexibility to make the cuts deeper by performing more search (at a cost) may be an advantage. The most important feature of the proposed method may well be the recursive partitioning process illustrated by the second numerical example. It clearly demonstrates that we can emulate the branch and cut algorithm without experiencing

the exponential growth of the need for memory space to store the branching information. However, the efficiency of this technique remains to be seen until extensive computational experimentations are done with it.

The new cuts may be used in conjuction with Gomory cuts and cuts generated using cover inequalities in the branch and cut process. When all cover inequality based cuts are exhausted, resorting to search based cuts before branching may be a viable alternative. Also, it may be possible use them interchangeably with Gomory cuts, using one when the other performs poorly. If there exists some complementarines, in this sense, then it may be possible to cut down the need for branching drastically in the branch and cut process.

The cover inequalities provide no cuts in the case of 0-1 programming problems with constraint matrices with all 0-1 entries. Set covering problems are good examples of such problems. The problems described by Corneujols and Dawande are another class for which cover inequalities seem to perform very badly. The search based cuts may perform better for such problems because their generation has nothing to do with the structure of the constraint matrix.

Acknowledgements

The author wishes to thank C. A. Gelogullari for doing the programming and carrying out the computational experiments.

References

BALAS, E. 1972. Integer Programming and Convex Analysis: Intersection Cuts from Outer Polars, *Mathematical Programming* 2, 330-382.

BALAS, E. 1975. Facets of the Knapsack Polytop, *Mathematical Programming* 8, 146-164.

BALAS, E. AND E. ZEMEL. 1978. Facets of the Knapsack Polytop from Minimal Covers. *SIAM Journal on Applied Mathematics* 34, 119-148

BALAS, E., S. CERIA, AND G. CORNUEJOLS. 1996. Mixed 0-1 Programming by Lift-and-Project in a Branch-and-Cut Framework. *Management Science* 42, 1229-1246.

BALAS, E., S. CERIA, M. DAWANDE, F. MARGOT, AND G. PATAKI. 2001. OCTANE: A New Heuristic for Pure 0-1 Programs, *Operations Research* 49, 207-225.

BIXBY, R. E., S. CERIA, C. M. McZEAL, AND M. W. P. SAVELSBERGH. 1998. An Updated Mixed Integer Programming Library: MIPLIB 3. *Technical Report TR98-03, Rice University, 1998.*

BURDET, C. A. 1972. Enumerative Inequalities in Integer Programming. *Mathematical Programming* 2, 32-64.

CPLEX (1997). Linear Optimizer 5.0 with Mixed Integer & Barrier Solvers, Copyright

(c) ILOG 1997.

CORNUEJOLS, G. AND M. DAWANDE. 1999. A Class of Hard Small 0-1 Programs. *INFORMS Journal on Computing* 11, 205-210.

CROWDER, H. P., E. L. JOHNSON, AND M. W. PADBERG. 1983. Solving Large Scale Zero-One Linear Programming Problems. *Operations Research* 31, 803-834.

FREVILLE, A., AND G. PLATEAU. 1990. Hard 0-1 Multiknapsack Test Problems for Size Reduction Methods. *Investigation Operativa* 1, 251-270.

GARFINKEL, R. S., AND G. L. NEMHAUSER. 1972. *Integer Programming* , Wiley, New York.

GLOVER, F. 1972. Cut Search Methods in Integer Programming. *Mathematical Programming* 3, 86-100.

GOMORY, R. E. 1958. Outline of an Algorithm for Integer Solutions to Linear Programs. *Bulletin of the American Mathematical Society* 64, 275-278.

GU, Z., G. L. NEMHAUSER, AND M. W. P. SAVELSBERGH. 1998. Lifted Cover Inequalities for 0-1 Integer Programs: Computation. *INFORMS Journal on Computation* 10, 427-437.

HAMMER, P. L., E. L. JOHNSON, AND U. N. PELED. 1975. Facets of Regular 0-1 Polytopes. *Mathematical Programming* 8, 179-206.

NEMHAUSER, G. L., AND P. H. VANCE. 1994. Lifted Cover Facets of the 0-1 Knapsack Polytope with GUB Constraints. *Operations Research Letters* 16, 255-263.

NEMHAUSER, G. L., AND L. A. WOLSEY. 1988. *Integer and Combinatorial Optimization* , Wiley, New York.

PADBERG, M. W., AND G. RINALDI. 1987. Optimization of a 532 City Symmetric Traveling Salesman Problem. *Operations Research Letters* 6, 1-7.

PETERSEN, C. C. 1967. Computational Experience with Variants of the Balas Algorithm Applied to the Selction of R&D Projects. *Management Science* 13, 736-750.

SHI, W. 1979. A Branch and Bound Method for the multiconstaint Zero-D One Knapsack Problem. *Journal of the Operational Research Society* 30, 369-378.

WEINGARTNER, H. M., AND D. N. NESS. 1967. Methods for the Solution of the Multidimensional 0/1 Knapsack Problem. *Operations Research* 15, 83-103.

WOLSEY, L. A. 1976. Facets and Strong Valid Inequalities for Integer Programs. *Operations Research* 24, 367-372.

WOLSEY, L. A. 1998. *Integer Programming* , Wiley, New York.

ZEMEL, E. 1978. Lifting the Facets of Zero-One Polytopes. *Mathematical Programming* 15, 268-277.