Solving Mixed Integer Linear Programs Using Branch and Cut Algorithm

by Shon Albert

A Project Submitted to the Graduate
Faculty of North Carolina State University
in Partial Fulfillment of the
Requirements for the Degree of
MASTERS OF MATHEMATCS
Major Subject: Mathematics

Acknowledgment

I would like to thank Dr. Kartik Sivaramakrishnan for his assistance with the project. His knowledge, time, and Branch and Bound code were pertinent to my research.

ABSTRACT

Mixed Integers Linear Programs maximize a linear objective function subject to linear inequalities and integrality constraints on some of the variables. This thesis studies how Mixed Integer Linear Programs are solved using a computer implementation of a Branch and Cut Algorithm. A Branch and Cut Algorithm combines the advantages of a pure Branch and Bound Scheme and Gomory Cutting Planes Scheme. Branch and Cut should be faster than Branch and Bound alone. We illustrate the effectiveness of the algorithm with a couple test examples.

CHAPTER 1
Mixed Integer Linear Programming

1.1 Introduction

Linear Programming (LP) is concerned with maximizing or minimizing an equation over certain criteria or constraints. One simple example might look like the following:

$$\begin{aligned} \max : z = \quad & 3x_1 - x_2 + 5x_3 & & \\ \text{such that} \quad & x_1 + 2x_2 - 4x_3 & \leq & \ 7 \\ & 7x_1 - 5x_2 - x_3 & \leq & \ 2 \\ & -x_1 + 3x_2 + 9x_3 & \leq & \ 13 \\ & x_i & \geq & \ 0, \ \ i = 1, \ldots, 3 \end{aligned} \tag{1.1}$$

This problem can be solved using the Revised Simplex Method (see section 1.7) or Dual Simplex Method (see section 1.8) to find a solution of:

$$x = (x_1, x_2, x_3) = (0.5, 0, 1.5)$$

Problems like these are used throughout businesses to minimize waste, maximize profit, or optimize other aspects of business. However some parts of the solution may be restricted to integers, $x \in Z^+$. Mixed Integer Linear Programs are LP's which require some or all of the variables to be integers. In example (1.1), if x_3 represented cases of product sold, this would present a problem since a company can't ship partial cases. Simply rounding the solution will often comprimise the optimality and the rounded solution may not be in the feasible region.

One way to solve MILPs is to enumerate all the integer solutions in the feasible region and individually check each one for optimality. However as the dimensions of the problem, n, grows, enumeration becomes NP hard, meaning the number of feasible integer solutions grow exponentially. Instead of growing by order of some

polynomial, $p(x) = ax^n + bx^{n+1} + \ldots + ex^0$, the enumeration will grow by order $p(x) = s^x$ for some function s. For more information on enumeration of solutions see Wolsey [3] section 1.4.

Even though MILPs can not be solved simply using the Revised or Dual Simplex Method both of these methods must be used in the process of solving MILPs.

1.2 Techniques for Solving MILP

MILPs are very hard to solve, however there are two methods we can use that will return $x \in Z^+$. One method is involves using Cutting Planes. The goal of cutting planes is to "cut" out a chunk of the feasible region based on information from the optimal LP dictionary. The other technique is called Branch and Bound, which is a recursive process using intelligent numeration

1.3 Contribution of the Project

While Cutting Planes can be solved very fast, the process can be unreliable. On the other hand Branch and Bound is very reliable but can be extremely slow. Our contribution to the subject is to show that combining these two techniques to solve large instance MILPs is more efficient than using solely Branch and Bound. The goal is to first make a couple of "cuts" using Gomory Cutting Planes, then solve the remaining MILP using Branch and Bound. If done correctly in large instances, this algorithm should be more efficient than Branch and Bound alone.

1.4 Solving MILP Using Branch and Bound

The problem with solving MILPs using the Revised Simplex Method is that it often returns parts of $x = (x_1, x_2, \ldots x_n)$ that are not in integer form as is the case in the earlier example(1.1). There are several methods we can use to find $x \in Z^+$, one of which is called the Branch and Bound Method. To solve using Branch and Bound we first solve the problem with Relaxation, which means we solve it using the Revised Simplex as if there were no integer restrictions. From this we obtain z:

$$z = cx$$

This is an upperbound on the MILP since we are maximizing over a larger set that encompasses all the integer solutions. Next we pick a noninteger variable and branch on it. Most commonly we branch on the most fractional variable, the one closest to half way between its floor and ceiling, or furthest from a whole integer. If branching on most fractional we pick $\hat{x_j}$:

$$\hat{x_j} \quad \text{such that} \quad j = \min[i : |\tfrac{1}{2} - (x(i) - \lfloor x(i) \rfloor)|] \quad \text{for} \quad i = 1 \ldots n$$

For example if

$$x = (1.25, 3.45, 2.75, 5)$$

We would branch on x_2 because .45 is closer to $\frac{1}{2}$ than (.25, .75, 0). By branching on a variable, we take the root problem and create two subproblems, or nodes. In this example , In one node we add the constraint: $x_2 \leq 3$. In the other node we add the constraint $x_2 \geq 4$. Let S be the feasible solution set of the LP relaxation: $S = \{x : Ax \leq b \text{ and } x \geq 0\}$, we now have:

$$\begin{aligned} S_1 &= S \cap \{x : x_j \leq \lfloor \hat{x_j} \rfloor\} \\ S_2 &= S \cap \{x : x_j \geq \lceil \hat{x_j} \rceil\} \end{aligned}$$

From here each of the branches must be solved and and checked for optimality. After solving the original system and branching on the first variable, we set bounds $\underline{z} = -\infty$ and $\overline{z} = c^T x$ where x is optimal in LP relaxation. Each created node is eventually either branched or pruned. A node can be pruned in the following different ways, optimality, bounds, and infeasibilty. We continue to solve the problem in the following manner:

1.4.1 STEP 1: Infeasibility

Pick a node and solve it with the Dual Simplex Method with updated constraints. If the Dual Simplex is unbounded then the problem is infeasible and the node is prunned by infeasibility. This occurs when the new constraint disagrees with

constraints already established.

1.4.2 STEP 2: Optimality/Bounds

We now check for bounds and optimality. If $z = cx$ for x, the solution to the Dual Simplex and $z > \underline{z}$ then we set $\underline{z} = z$, and this is our new lowerbound. This node is now pruned by optimality since it is currently the best solution to the MILP. If $z \leq \underline{z}$ then this is worse than our lowerbound, and this node is prunned by bounds.

1.4.3 STEP 3 : Branching

If x is not in integer form then we branch on the variable that is most fractional.

1.4.4 STEP 4 : Repeat

Pick a new node and repeat starting on STEP 1, Until all nodes are pruned, at which time the node associated with the lowerbound $\underline{z}$ is optimal.

1.5 Example

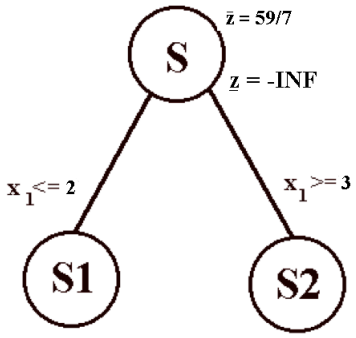
For illustration, consider the following problem

$$\begin{array}{rrcl} \max z = & 4x_1 - x_2 & & \\ \text{s.t.} & 7x_1 - 2x_2 & \leq & 14 \\ & 0x_1 + x_2 & \leq & 3 \\ & 2x_1 - 2x_2 & \leq & 3 \\ & x_i & \in Z^+ & 0, \;\; i = 1, 2 \end{array} \tag{1.2}$$

Let S be the feasible solution set of the LP relaxation: $S = \{x : Ax \leq b \text{ and } x \geq 0\}$. Solving the LP relaxation using the Revised Simplex Method, we obtain $\overline{z} = \frac{59}{7}$ with $x = (\frac{20}{7}, 3)$. We set $\underline{z} = -\infty$. Since only x_1 is fractional this is the variable that we branch on. Now we have:

$$\begin{array}{rcccl} S_1 & = & S \cap \{x : x_1 \leq \lfloor \frac{20}{7} \rfloor\} & = & S \cap \{x : x_1 \leq 2\} \\ S_2 & = & S \cap \{x : x_1 \geq \lceil \frac{20}{7} \rceil\} & = & S \cap \{x : x_1 \geq 3\} \end{array}$$

And our branch tree look like this:



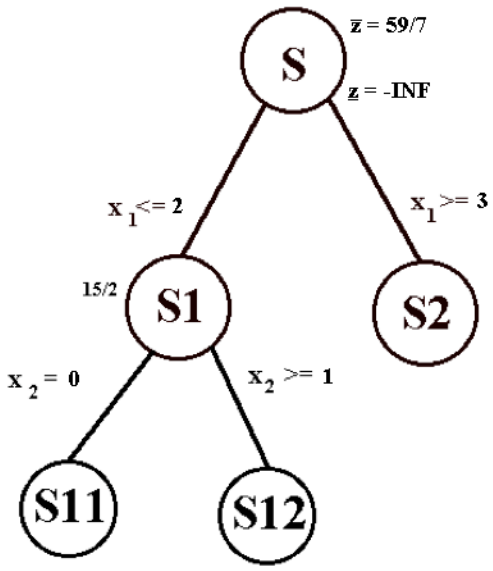
Now we have to choose a node to work on first. Pick S_1. We add the constraint $x_1 \leq 2$ to the system S.

$$S_1 = S \cap \{x : x_1 \leq 2\}$$

After two iterations in the Dual Simplex Method we get $x = (2, \frac{1}{2})$. Since this is not in integer form we once again branch, this time on x_2. so we have:

$$\begin{aligned} S_{11} &= S_1 \cap \{x : x_2 = 0\} \\ S_{12} &= S_1 \cap \{x : x_2 \geq 1\} \end{aligned}$$

And our tree looks like

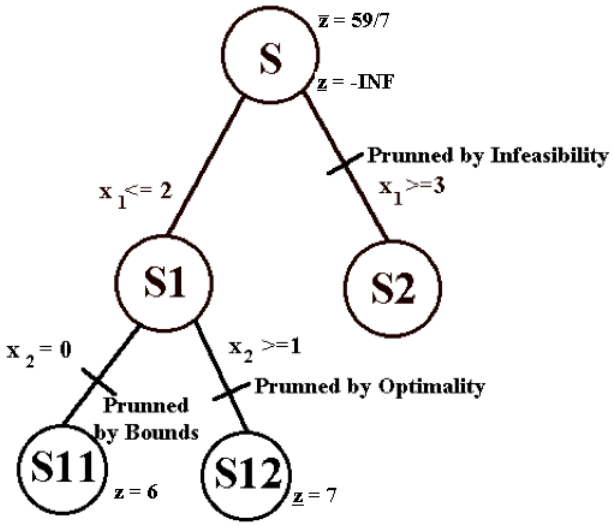


There are three nodes left and many strategies as to how to pick them, arbitrarily we pick S_2. We add the constraint $x_1 \geq 3$. However we have a contradiction with the constraints. we know $7x_1 - 2x_2 \leq 14$, so $21 - 2x_2 \leq 14$ since $21 - 2x_2 \leq 7x_1 - 2x_2 \leq 14$ this implies that $x_2 \geq \frac{7}{2}$, which contradicts the second constraint of the problem: $x_2 \leq 3$. The Dual Simplex Method returns the problem as infeasible and we can prune this node by infeasibility.

The two nodes remaining are S_{11} and S_{12}. Choosing S_{12}, the DSM returns an optimal solution of $x = (2, 1)$ and z = 7. We compare z to $\underline{z}$. Since $z > \underline{z}$ this node is prunned by optimality and we now set $\underline{z} = 7$ and pick a new node.

The only remaining node is S_{11}. DSM returns $x = (\frac{3}{2}, 0)$ with $z = 6$. Since $z < \underline{z}$ this node is prunned by bounds.

The node list is now empty and the problem is finished with the optimal solution of $x = (2, 1)$ and z = 7. Our final branch tree is:



1.6 Gomory Cutting Planes

Adding Cutting Planes to the system is another good way to solve MILPs. Cutting Planes are very fast but can be an unstable method(see Wolsey [3] chapter 8). The idea of cutting planes is cut out chunks of the feasible region based on information from the optimal dictionary. First we solve the relaxation in the RSM. We now look at the information from the optimal dictionary and from here we are able to deduce an inequality that will "cut" out a piece of the feasible region, then we add the new cut to the system and reoptimize using the DSM. This process is repeated until the required variables are integers.

One specific type of Cutting Plane is Gomory Cuts, which are made in the following manner if $v = x_{B(i)}$ is basic and noninteger then we look at its information in the optimal dictionary:

$$x_{B(i)} + bx_{N(j)} + cx_{N(k)} + \ldots + ex_{N(l)} = v$$

All x_N are nonbasic. Since v is noninteger we know that: $x_{B(i)} \leq v$ so $x_{B(i)} \leq \lfloor v \rfloor$ so:

$$bx_{N(j)} + cx_{N(k)} + \ldots + ex_{N(l)} \geq v - \lfloor v \rfloor$$

We can add this inequality to the system and reoptimize in DSM. Repeat until $x \in Z^+$ For illustration consider the same example using Gomory Cuts:

1.6.1 Example

$$\begin{array}{rrrll} \max z = & 4x_1 - x_2 & & & \\ \text{s.t.} & 7x_1 - 2x_2 & \leq & 14 & \\ & x_2 & \leq & 3 & \\ & 2x_1 - 2x_2 & \leq & 3 & \\ & x_i & \in Z^+ & 0, \ \ i = 1, 2 & \end{array} \tag{1.3}$$

After solving the relaxation in the RSM we get the solution optimal solution $x = (\frac{20}{7}, 3, 0, 0, \frac{23}{7})$. The optimal dictionary is:

$$\begin{array}{rrrrrrr} x_1 & & +\frac{1}{3}x_3 & +\frac{2}{7}x_4 & & = & \frac{20}{7} \\ & x_2 & & +x_4 & & = & 3 \\ & & -\frac{2}{7}x_3 & +\frac{10}{7}x_4 & +x_5 & = & \frac{23}{7} \end{array}$$

Since $x_1 + \frac{1}{3}x_3 + \frac{2}{7}x_4 = \frac{20}{7}$ is fractional, we add the cut: $\frac{1}{3}x_3 + \frac{2}{7}x_4 \geq \frac{6}{7}$, and call it s:

$$s = -\frac{6}{7} + \frac{1}{3}x_3 + \frac{2}{7}x_4$$

After adding this cut to the system$\{c, A, b, x\}$, we reoptimize using the DSM which returns $x = (2, \frac{1}{2}, 1, \frac{5}{2}, 0)$, since x_2 is noninteger we look at the dictionary.

$$\begin{array}{rrrrrrr} x_1 & & & & +s & = & 2 \\ & x_2 & & -\frac{1}{5}x_5 & +s & = & \frac{1}{2} \\ & & x_3 & -x_5 & -5s & = & 1 \\ & & & x_4 \quad +\frac{1}{2}x_5 & +6s & = & \frac{5}{2} \end{array}$$

We now add the cut based on information in row 2 since we are dealing with x_2. We add the cut $\frac{1}{2}x_5 \geq \frac{1}{2}$ or $t = -\frac{1}{2} + \frac{1}{2}x_5$. After reoptimizing in the DSM we

arrive at the solution $x = (2, 1, 2, 2, 1, 0, 0)$. This is completely integer and hence optimal for the MILP. Notice that $\overline{x} = (x_1, x_2) = (2, 1)$ is the same as when we solved this MILP with branch and bound.

An advantage of Gomory Cutting Planes is they can produce solutions quickly on the computer without using a lot of memory. A disadvantage is large instances can can cause the algorithm to get stuck making a series of almost identical miniscual cuts, and can take a large number of iterations to solve. This will cause the system to become unstable.

1.7 Revised Simplex Method

In order to solve the LP relaxation we must rely on the Revised Simplex Method, the algorithm of choice for solving ordinary LPs. RSM is conducted in the following manner:

1.7.1 STEP 1:Initialization

The first step is to initialize the problem. We begin by introducing slack variables.

$$\text{If} \quad ax_1 + bx_2 + cx_3 \quad \leq \quad d$$

then there exists a slack variable $x_4 \geq 0$ such that

$$ax_1 + bx_2 + cx_3 + x_4 \quad = \quad d$$

so

$$x_4 \quad = \quad d - ax_1 - bx_2 - cx_3$$

We add a slack variable for every constraint in the problem. For $\mathrm{A}_{m \times n}$ there will be m new slack variables added to x and m new columns of A. After this $\mathrm{A}x \leq b$ can be rewritten:

$$[A|I_{m\times m}] \begin{bmatrix} x_1 \\ \vdots \\ x_n \\ x_{n+1} \\ \vdots \\ x_{n+m} \end{bmatrix} = [b]$$

c must also grow to account for the new variables in x, so we append m zeros to the tail of c since z will not depend on the slack variables.

$$c = (c_1 \dots c_n, 0 \dots 0)$$

The Revised Simplex method needs a starting solution. Any x such that $\mathrm{A}x = b$ works as long as $x \geq 0$. An easy way to get a basic feasible solution is to set the slack variables equal to b:

$$x_0 = (0_1 \dots 0_m, b_1 \dots b_n)$$

This always satisfies $\mathrm{A}x = b$. We now seperate A into [A_B A_N], where A_B is the columns of A corresponding to what are called the basic variables. Basic variables are the parts of x not given to be zero. A_N is the columns of A corresponding to the non basic variables. In the first iteration the basic variables are the slack variables, $(x_{n+1}, \dots x_{n+m})$ since everything else in x is initialized to 0. So A_B=[$\mathrm{A}_{n+1} \dots \mathrm{A}_{n+m}$] and A_N=[$\mathrm{A}_1 \dots \mathrm{A}_n$] and still $\mathrm{A}x$=$\mathrm{A}_B x_B$ + $\mathrm{A}_N x_N$ = b.

1.7.2 STEP 2

Solve for

$$A_B^T y = c_B$$

then compute

$$S_N = C_N - A_N^T y$$

If $S_N \leq 0$ then we declare optimality and are finished. If $S_N(i) > 0$ for any i then we can improve the solution, z, by increasing one of the values of x. If i is the index equal to max S_N(i) then $x(i)$ acts as a pivot variable and changing it will have the most increasing affect on z. $x(i) \in x_N$ is called the entering nonbasic variable.

1.7.3 STEP 3

Solve $A_B d = A_N$ for d, then compute $\theta = \min [x_B(i)/d(i)]$ such that d(i) > 0. Let

$$\theta \quad = \quad x_B(\ell)/d(\ell)$$

then $x_B(\ell)$ is the leaving basic variable. This must be switched with the entering nonbasic variable. By switching the nonbasic and basic variable, we, in a sense, slide the solution x by decreasing $x_{B(\ell)}$ to zero, and putting the slack created by this into $S_{N(i)}$. By doing so z is increased.

1.7.4 STEP 4

Now we update B and N, the list of basic and nonbasic variables, and then update x_B in the following manner:

$$x_B \quad = \quad x_B - \theta d$$

and

$$x_B(\ell) \quad = \quad \theta$$

1.7.5 Repeat

Now go back to STEP 2 and repeat until $S_N \leq 0$

When $S_N \leq 0$ we are optimal and the solution $\overline{x} = (x_1, x_2, \ldots x_n)$ where $\overline{x}$ is the first n variables of x, since the last m are slack and not required to be integer.

1.8 DUALITY

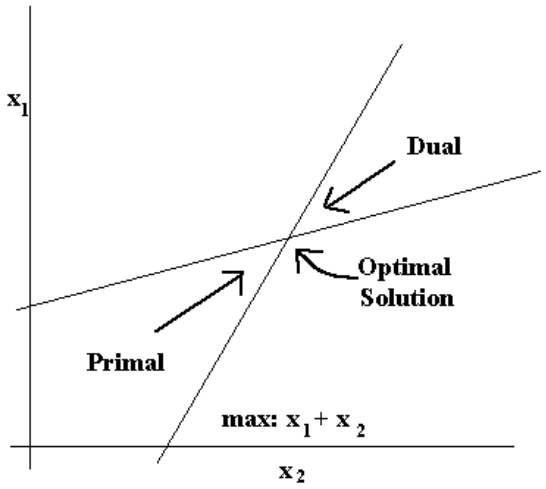
The problem in standard form is: max $z = cx$ such that A$x \leq b$ for x nonnegative. This is called the Primal LP. The Dual of the LP is motivated by finding an upperbound on the objective function. In order to solve the Dual we must invert the system. To do this we exchange the roles of the objective function and the right hand side constraints. A is transposed and instead of using primal variable x we use dual variable y. The dual is a minimization problem. The Primal LP becomes the Dual in the following manner:

$$\begin{array}{lll} \max & c^T x & \\ s.t. & Ax \leq b & \end{array} \quad \Longrightarrow \quad \begin{array}{lll} \min & b^T y & \\ s.t. & A^T y \geq c & \end{array}$$

Our first example would change in the following manner:

$$\begin{array}{lrcl} \max & 3x_1 - x_2 + 5x_3 & & \\ \text{s.t.} & x_1 + 2x_2 - 4x_3 & \leq & 7 \\ & 7x_1 - 5x_2 - x_3 & \leq & 2 \\ & -x_1 + 3x_2 + 9x_3 & \leq & 13 \end{array} \quad \Longrightarrow \quad \begin{array}{lrcl} \min & 7y_1 + 2y_2 + 13y_3 & & \\ \text{s.t.} & y_1 + 7y_2 - 1y_3 & \geq & 3 \\ & 2y_1 - 5y_2 + 3y_3 & \geq & -1 \\ & -4y_1 - y_2 + 9y_3 & \geq & 5 \end{array} \tag{1.4}$$

All solutions to the dual are upperbounds on the primal. We minimize the dual to find the least upperbound. A two dimensional problem might look like this:



The primal and the dual arrive at the same solution, the dual just takes a "backdoor" method. The same way we can solve the problem using the Revised Simplex Method, we have the Dual Simplex Method. DSM is solved in a similar fashion to RSM. Steps to solve using the DSM can be found on pages(152-157) of Chvatal[1].

CHAPTER 2
Research and Results

The goal of the project is to illustrate that combining Gomory Cutting Planes with the Branch and Bound method is faster than doing Branch and Bound alone. While Cutting Planes are fast they often get stuck causing the method to be unstable. Branch and Bound is stable but can take up to 2^n nodes to complete. Finding a more efficient means can prove to be valuable by saving the user time and the computer space.

2.1 Schematic for Branch and Cut algorithm

In order to solve large order MILP problems I implemented 2 algorithms in MATLAB. Branch and Bound code was provided by Dr K. Sivaramakrishnan. To complement his code, I implemented the Gomory cutting scheme earlier described. The second code takes the original system S = $\{c, A, b, x\}$ and passes through a limited number of Gomory Cuts and gets $S_G = \{c_G, A_G, b_G, x_G\}$ then passes this to Branch and Bound to finish solving the problem.

2.2 Implementation

The Branch and Bound code solved all problems without error. However, large instance took a large number of nodes to complete. In order to compare the results I added a counter to indicate how many nodes were created. Since the Branch and Bound program is run recursively, the counter had to be implemented the same way. I added a command in each node that would return the following number to its root node:

$$\begin{aligned} 1 &: \text{If Node is prunned} \\ \textstyle\sum\text{Left node} + \textstyle\sum\text{Right node} + 1 &: \text{If Node is branched} \end{aligned}$$

I also implemented the Gomory Cutting Scheme (described section 1.6). Adding a counter in the Cutting Plane code was easier since the program just repeats

in a while loop. To allow the program to coincide with Branch and Bound, I added a command to break out of the program if $iter \geq itermax$ for some variable defined by the user $itermax$, and $iter$ the number of cuts. Instead of the code returning just x and optimal solution z, it returns S_G so the updated system can be used by branch and bound to finish solving the problem. The code used to implement the branch and cut scheme can be found at the following website `http://www4.ncsu.edu/~kksivara/shon-codes`

2.3 Computational Results

The results were encouraging. In most of the large order problems running through Cut and Branch created less nodes in Branch and Bound than using Branch and Bound alone. Look at the following results

2.3.1 Example 1

$$\begin{aligned} \max z = \quad & 4x_1 - x_2 & & \\ \text{s.t.} \quad & 7x_1 - 2x_2 & \leq \quad & 14 \\ & x_2 & \leq \quad & 3 \\ & 2x_1 - 2x_2 & \leq \quad & 3 \\ & x_i & \in Z^+ \quad & 0, \;\; i = 1, 2 \end{aligned} \tag{2.1}$$

When we solve this example using Branch and Cut we get the following results. (Cuts is number of iterations in the Cutting Plane method, Nodes is the number of nodes to complete Branch and Bound, and Total Nodes is the sum of Cuts and Nodes)

$\underline{Cuts}$	$\underline{Nodes}$	$\underline{TotalNodes}$
0	7	7
1	5	6

As we see, adding just one Cutting Plane reduces the number of nodes to solve in Branch and Bound from 7 to 5. While 2 nodes may seem miniscule it is still substantial considering that one cut reduced the number of node by 28 percent.

2.3.2 Example 2

Let n be odd.

$$\begin{array}{lrcl} \max & z = -x_{n+1} & & \\ \text{s.t.:} & 2x_1 + 2x_2 + \ldots + 2x_n + x_{n+1} & \leq & n \end{array}$$

Look at the following results when n=9.

$\underline{Cuts}$	$\underline{Nodes}$	$\underline{TotalNodes}$
0	251	251
1	69	70
2	19	21
3	5	8

Pure Branch and Bound created a total of 251 nodes to solve. Even if it takes very little time to solve each node, there are a lot to solve. As we add in a cutting planes, the number of total number of nodes to solve the problem decreases rapidly.

2.3.3 Example 3

Let n be odd.

$$\begin{array}{lrcl} \max & -x_{n+1} & & \\ \text{s.t.:} & 2x_1 + 2x_2 + \ldots + 2x_n + x_{n+1} & \leq & n \end{array}$$

This time let n be any odd integer. The following results show that as n grows, the number of nodes created to solve using Branch and Bound grows faster. (BaB is number of nodes in pure Branch and Bound, "x"cuts is the number of nodes created in Branch and Bound after making x Gomory Cuts)

$n =$	BaB	$1Cut$	$2Cuts$	$3Cuts$	$4Cuts$
3	5	1			
5	19	5	1		
7	69	19	5	1	
9	251	69	19	5	1
11	923	251	69	19	5
13	3431	923	251	69	19
15	12869	3431	923	251	69

Notice that as n grows the number of nodes in Branch and Bound grows by order of 2^n. While it may take a very short amount of time to solve a node in BaB, having 2^n nodes to solve will can take quite a while. Not only did adding cutting planes decrease the number of nodes in BaB, it equivalently reduced the problem to to a smaller one. If n=11 then it takes 923 nodes to solve. However adding one cutting plane reduces the number of nodes to 251, the exact number of nodes used to solve n=9 in pure Branch and Bound. What we see it that by adding x cutting planes, we are essentially reducing the number of nodes by order 2^{2x} in this example.

2.4 Conclusion

Gomory Cutting Planes are fast, but unreliable. Branch and Bound is reliable, but slow. We can combine the two algorithms into one, called Branch and Cut. If we first take take a few cuts using the Gomory Scheme then apply the remaining system to Branch and Bound, we have an algorithm that is not only reliable, but faster than Branch and Bound alone. In the examples illustrated we find that taking x Gomery Cuts can essentially reduce the number of nodes created drastically. Ultimately we find that Branch and Cut is a more efficient algorithm than Branch and Bound.

LITERATURE CITED

[1] Vasek Chvátal, *Linear Programming*, W.H. Freeman and Company, 1983.

[2] Kartik Sivaramakrishnan, *MA/OR/IE 505 - Linear Programming*, Spring 2006, North Carolina State University. Available at `http://www4.ncsu.edu/~kksivara/ma505`

[3] Laurence Wolsey, *Integer Programming*, John Wiley and Sons, Inc, 1998