2.1 Introduction

From many years, research has been going on to solve the Integer Programming Problems; also there are several algorithms available for solving the Integer Programming Problems. However, most techniques can be classified either Enumeration Techniques given by Land and Doig (1960), Balas (1963), Dakin (1965), Lawler and Wood (1966), or Cutting Plane methods by Dantzig (1959), Gomory (1963), Yong (1968). Enumerations techniques are designed to exploit the fact that the feasible region of a branched integer program always contains a finite subset of feasible points. Branch and Bound, and Implicit Enumeration are the techniques where one attempt to enumerate only a small subset of the feasible integer points, while concluding that the remaining points are inferior to those examined.

If an optimal solution to a linear programming problem exists, the simplex algorithm always finds an extreme point optimum.
This is the basic motivation for cutting plane methods. Constraints (or cutting planes) are successively added to the linear programming relaxation of an integer-programming problem in such a way that the current non-integer optimal extreme points is cut away or made infeasible. However, this is done so that the entire set of integer points remains feasible. By proceeding in this manner, a new convex set is constructed that eventually has an integer point as an extreme point Gomary (1963). Thus, an optimal integer solution can be found by solving a sequence of linear programs.

In this technique a new type of cut is added to the problem after finding the solution to the linear programming relaxation problem. This cut is derived by finding the minimum difference from the integer points, which are inside the feasible region to the objective surface passing through this point and parallel to the objective function surface. The cut has been designed in such a way that the total number of integer solutions in the resulting feasible region is substantially reduced. After adding A-T cut (2005) we can obtain the integer optimum.
2.2 Derivation of the Cut

Consider the following pure Integer Programming Problem

\[
\begin{align*}
\text{Max } Z &= c^T x \\
\text{s.t. } A_{m \times n} x &\leq b, \\
x &\geq 0 \text{ and integer}
\end{align*}
\]

and the linear programming relaxation associated with this problem is given by

\[
\begin{align*}
\text{Max } Z &= c^T x \\
\text{s.t. } A_{m \times n} x &\leq b, \\
x &\geq 0
\end{align*}
\]

First, we solve the linear programming relaxation using any LP method.

Let the solution be \( x^* \). If \( x^* \) is all integer, then the problem is solved. Otherwise,

Let the \( k^{th} \) component of \( x^* \) be non-integer with \( x^k = b^*_k \).

The nearest integer values to \( x^k \) be

\[
x_1^k = [b^*_k] \quad \text{and} \quad x_2^k = [b^*_k] + 1 = \{b^*_k\}, \quad \text{for } k = 1, 2, \ldots, n
\]

where \([t]\) is the largest integer less than or equal to \( t \) and \( \{t\} \) is the smallest integer greater than or equal to \( t \). With such bifurcations we can find all the \( 2^n \) points in the surrounding of the non-integer solution \( x^* \).
Denote the set of indices of these $2^n$ points by $T$.

Let the objective value at $x^*$ be $Z^*$. Thus, the objective function level plane at $x^*$ will be $cx^* = Z^*$.

Now we find the difference $d_i = Z^* - cx_i^0$, $i \in T$, i.e., the difference between the objective function value at non-integer solution and the objective function values at surrounding integer points, instead of using the formula of perpendicular distance discussed by Bari and Shoeb (2003).

Where $x_i^0$, $i \in T$ are surrounding integer points around $x^*$.

Now, we search for the feasible point $x_i^0$, which has a minimum positive difference from the objective function value. Obviously the negative difference and the differences from the infeasible points should be omitted.

Let $S$ be the set of indices $i \in T$ for which, $x_i^0$'s are feasible.

Let

$$x^0 = \left\{ x_k^0 \bigg| d_k = \min_{i \in S} d_i \right\}$$

A plane passing through this integer point and parallel to the objective hyper-plane will be $cx^0 = Z^0$.

Clearly, $Z^0 < Z^*$. 
We will introduce the cut as
\[ cx^0 \geq Z^0. \]

This cut is nothing but same as NAZ cut, developed by Bari and Shoeb (2003).

Where, \( Z^0 \) acts as a lower bound for the integer solution to the problem (2.2).

To find the integer optimum solution we add the A-T cut due to Bari and Teg (2005) at \( x^0 \) as
\[
\sum_{j=1}^{n} x_j = \sum_{k \in S} \alpha_j^k, \quad \alpha \in x^0
\]

### 2.3 Numerical Illustration

Maximize \( Z = 2x_1 + 3x_2 \)

Subject to \begin{align*}
5x_1 + 2x_2 & \leq 15 \\
3x_1 + 5x_2 & \leq 15 \\
x_1, x_2 & \geq 0 \text{ and integer}
\end{align*}

After solving the linear programming relaxation problem by using any conventional Linear Programming method we get the non-integer optimal solution as:

\[ x_1 = 2.37, \quad x_2 = 1.58 \quad \text{and} \quad Z = 9.43. \]
So we round off the non-integer solution to the four nearest integer points i.e., (2,1), (2,2), (3,1), (3,2). The respective differences are

Difference from the point (2,1): $9.43 - 7 = 2.43$;
Difference from the point (2,2): $9.43 - 10 = -0.57$;
Difference from the point (3,1): $9.43 - 9 = 0.43$ and
Difference from the point (3,2): $9.43 - 12 = -2.57$

We are left with only one feasible point (2,1), which gives the minimum positive difference. Now the NAZ cut passing through the integer point (2,1) and parallel to the objective function is

$$2x_1 + 3x_2 \geq 7.$$  \hspace{1cm} (2.3)

To find the integer optimum we add the A-T cut passing through the integer point (2,1).

$$x_1 + x_2 = 3$$ \hspace{1cm} (2.4)

After adding these two cuts (2.3) and (2.4)

$$2x_1 + 3x_2 \geq 7$$
$$x_1 + x_2 = 3$$

We find the optimal integer solution as follows:
Iteration 1.

\[ x^1 = (1,2) \text{ with } Z^1 = 8 \]

Iteration 2.

\[ x^2 = (2,1) \text{ with } Z^2 = 7 \]

Iteration 3.

\[ x^3 = (0,3) \text{ with } Z^3 = 9 \]

Terminating point will be at \((0,3)\)

\[ x_1 = 0, \; x_2 = 3 \text{ and } Z = 9 \]

This problem may also be seen graphically.
2.4 Graphical Illustration

(0,3) Integer Optimal Solution

X'(2.37,1.58) Non-integer Optimal Solution

A-T Cut

NAZ Cut

x^0(2,1)
2. Iteration
=>X = (2 1)
ZL = 7

3. Iteration
=>X = (0 3)
ZL = 9
Terminating point will be at (0 3) with Z* = 9.
⇒ X1* = 0
⇒ X2* = 3.

The Computer Program is as follows:

```c
#include<conio.h>
#include<math.h>
#include<iostream.h>
#include<process.h>

//...FUNCTION DECLARATION.....
void createset();
void createcoordinate();
void checkmindiff();
void constraintssatisfy();
void arrange_z_opt();
void createsets();
void selectcoordinate();
void cal_difference();
void constraintssatisfys();

struct optzcord
{
    int oz;
    int oxc[10];
}ozc[32], ansozc;
struct difference
{
    float diff;
    int xc[10];
}diff[32];
```
2.5 Computer Program

The example (given in chapter II) was run on PENTIUM under Windows VERSION 98 operating system. The TURBO C++ version 5.0 was used for developing the program. The program can easily solve up to five variables problem and takes only integer coefficients of objective function and constraints for the problem (given in chapter II). The programming code (C++), \(x^0, x^1, x^2, \ldots\) are represented by \(X_0, X^1, X^2, \ldots\) respectively and \(Z^0, Z^1, Z^2, \ldots\) are represented by \(Z_L, Z_L^1, Z_L^2, \ldots\) respectively.

The computer program solves the numerical problem as follows:

\[
\text{<<-: OUTPUT:-->}
\]

1st Step: Minimum feasible difference at (2 1)
---------------------------------------------------------------

\(=>X_0=(2 1)\)
\(X_0^1=2\) \(X_0^2=1\) \(Z_L=7\)

\[
\text{<<<--: 2nd Step: Search for integer optimum: -->>>
}\]

---------------------------------------------------------------

1. Iteration

\(=>X=(1 2)\)
\(Z_L^1=8\)
//GLOBAL VARIABLES
int num,ob_fun_coeff[10],coeff_of_const[10][10],b[10],num_eq,sumx;
int x[32][10],y[10][2],iteration=1;
float x_opt[10],z_opt;

//MAIN PROGRAM
void main()
{
    int i,j,l,p,sum=0;
    ansocz.oz=0;
    clrscr;
    printf("the objective function is like :");
    printf("max Z=c1.x1+c2.x2+c3.x3+......");
    printf("enter the variables(max. limit=10)");
    scanf("%d",&num);
    for(i=0;i<num;i++)
    {
        printf("input the value of c%d(integer value) ",i+1);
        scanf("%d",&ob_fun_coeff[i]);
    }
    for(i=0;i<num;i++)
    {
        printf("input the LPR optimum value of x%d ",i+1);
        scanf("%f",&x_opt[i]);
        if(x_opt[i]!=x_opt[i])
        {
            y[i][0]=(int)x_opt[i];
            else
            y[i][0]=x_opt[i];
        }
        createset();
        createcoordinate();
        printf("enter the LP relaxation optimal value of objective zu=");
        scanf("%f",&z_opt);
        printf("***enter the constraints***");
        printf("\n the no. of constraints:");
        scanf("%d",&num_eq);
        for(i=0;i<num_eq;i++)
        {
            for(j=0;j<num;j++)
            {
printf("\ninputa%d%d=",i+1,j+1);
scanf("%d",&coeff_of_const[i][j]);
}
printf("\nb%d=",i+1);
scanf("%d",&b[i]);
}
cal_difference();
ckeckmindis();
constraintssatisfys();
clrscr();
printf("\n\n<<: output :>>\n");
cout<<"\n\n1st step :minimum feasible differences at ( ";
for(int f=0;f<num;f++)
cout<<ozc[0].oxc[f]<<" ";
cout<<")";
printf("\n\t------\n");
printf("\n\t=> xo=");
for(f=0;f<num;f++)
printf("xo%d=%d",f+1,ozc[0].oxc[f]);
printf("z%d=%d ",ozc[0].oz);
printf("\n\n<<: 2nd step:search for integer optimum:--->>>\n");
printf("\n\t--------------\n");
while(1)
{
if(ansozc.oz>ozc[0].oz||ozc[0].oz>=(int)z_opt)
break;
ansozc=ozc[0];
sum=0;
for(int i=0;i<num;i++)
sum+=ozc[0].oxc[i];
sumx=sum;
for(i=0;i<num;i++)
    y[i][0]=ozc[0].oxc[i]-1;
createsets();
createcoordinate();
selectcoordinate();
constraintssatisfys();
printf("\n\n%d. Iteration",iteration);
printf("\n\t=> Xo =");
for(int r=0;r<iteration;r++);
printf("*");
printf("= ( ");
for(int f=0;f<num;f++)
printf(" %d ",ozc[0].oxc[f]);
printf("\n\t");
printf("ZL");
for(r=0;r<iteration;r++)
printf(" ");
printf(" =%d",ozc[0].oz);
iteration++;
if(iteration==10)
break;
}
if(ansozc.oz>ozc[0].oz)
ozc[0]=ansozc;
printf("\n\n\tTerminating point will be at ( ");
for(i=0;i<num;i++)
printf("%d",ozc[0].oxc[i]);
printf("\n\t");
printf(" with Z*=%d",ozc[0].oz);
printf("\n\t");
for(i=0;i<num;i++)
printf(" => X%d*=% d",i+1,ozc[0].oxc[i]);
printf("--------------------------------------");
getch();
}

//FUNCTIONS FOR CREATE SETS OF COORDINATE...
void createset()
{
for(int i=0;i<num;i++)
y[i][i]=y[i][0]+1;
}

//FUNCTION FOR CREATE COORDINATE SET....
void createcoordinate()
{
int count,k;
for(int i=0;i<num;i++)
{
count=0;
k=0;
    for(int j=0;j<pow(2,num);j++)
    {
        if(k<pow(2,num)/pow(2,i+l))
        {
            x[j][i]=y[i][0];
k++;}
    else
    {
        count++;
x[j][i]=y[i][1];
k++;
        if(count==pow(2,num)/pow(2,i+l))
        {
            k=0;
count=0;
        }
    }
//FUNCTION FOR CALCULATION OF DIFFERENCE......
void cal_difference()
{
    float c_val;
it z_val=0;
it sum=0;
float min_diff=0.0;
for(int i=0;i<pow(2,num);i++)
{ c_val=0;
sum=0;
z_val=0;
for(int j=0;j<num;j++)
{
diff[i].xc[j]=x[i][j];
z_val+=x[i][j]*ob_fun_coeff[j];
sum+=ob_fun_coeff[j]*ob_fun_coeff[j];
}
min_diff=(z_opt-z_val);
diff[i].diff=min_diff;
}
}

// FUNCTION FOR CHECKING OF MINIMUM DIFFERENCE
void checkmindiff()
{
    struct difference sd;
    for(int i=0;i<pow(2,num);i++)
    {
        for(int j=0;j<pow(2,num)-i-1;j++)
        {
            if((diff[j].dis<=0.0 & diff[j+l].diff>=0.0) || (diff[i].diff>diff[j+l].diff))
            {
                sd=diff[i];
                diff[i]=diff[i+1]
                diff[i+1]=sd;
            }
        }
    }
}

// FUNCTION FOR CALCULATION OF OPTIMUM Z VALUE BY
// SATISFYING CONSTRAINTS FOR 1st ITERATION
void constraints_satisfy()
{
    int z_val=0,sum=0,flag[10];
    int(i=0;i<pow(2,num);i++)
    {
        for(int k=0;k<num_eq;k++)
        {
            sum=0;
            for(int j=0;j<num;j++)
                sum+=coeff_of_const[k][j]*x[i][j];
            if(sum<=b[k])
                flag[k]=1;
            else
                flag[k]=0;
        }
int var=0;
for(int q=0;q<num_eq;q++)
if(flag[q]==1)
var++;
if(var==num_eq)
{
    z_val=0;
    for(int l=0;l<num;l++)
    {
        z_val+=ob_fun_coeff[l]*x[i][l];
    }
    ozc[i].oz=z_val;
}
else
{
    ozc[i].oz=0;
}
for(int m=0;m<num;m++)
ozc[i].oxc[m]=x[i][m];
}
arrange_z_opt();
}
//FUNCTION FOR ARRENGING OPTIMAL Z VALUE......
void arrange_z_opt()
{
struct optzcord opzcd;
for(int i=0;i<pow(2,num);i++)
{
    for(int j=0;j<pow(2,num)-i-1;j++)
    {
        opzcd=ozc[j];
        ozc[j]=ozc[j+1];
        ozc[j+1]=opzcd;
    }
}
}
//FUNCTION FOR CREATE SET OF COORDINATE FOR 2ND 3RD ITERATION
void createsets();
{ 
  for(int i=0;i<num;i++)
  y[i][1]=y[i][0]+2;
}

//FUNCTION FOR SELECT COORDINATE SET FOR 2nd 3rd ITERATION
void selectcoordinate()
{
  int sum1=0, sum2=0, xx[10][10], i=0;
  sum1=sumx;
  for(int j=0;j<pow(2,num);j++)
  {
    sum2=0;
    for(int k=0;k<num;k++)
      if(x[j][k]>=0)
        sum2+=x[j][k];
    if(sum==sum2)
      {
        for(int l=0;l<num;l++)
          xx[i][l]=x[j][l];
        i++;
      }
  }
  for(int m=0;m<i;m++)
  {
    for(int s=0;s<num;s++)
      x[m][s]=xx[m][s];
  }
  for(int a=i;a<pow(2,num);a++)
  {
   for(int c=0;c<num;c++)
    x[a][c]=0;
  }
}