CHAPTER II

INTEGER PROGRAMMING PROBLEM AND ALGORITHMS

2.1 Introduction

In this chapter the various techniques available to solve Linear Integer Programming problems are to be discussed in detail. The most popular methods namely Cutting Plane, Branch and Bound, Branch and Cut & Branch and Price are to be illustrated. Also the algorithms for each methods and drawbacks of each are to be discussed.

2.2 All Integer Programming Problems [Ham04] [Sha04]

A systematic procedure called Gomory's all integer algorithm discussed here generates 'cuts' (additional linear constraints) so as to ensure an integer solution to the given linear programming problem in a finite number of steps. Gomory's algorithm has the following properties.

1. Additional linear constraints never cut-off that portion of the original feasible solution space which contains a feasible integer solution to the original problem.
2. Each new additional constraint (or hyperplane) cuts-off the current non-integer optimal solution to the linear programming problem.

2.2.1 Method for Constructing Additional Constraint (Cut)

Gomory's method begins by solving the linear programming problem without taking into consideration the integer value requirement of the decision variables. If the solution so obtained is an integer, i.e. all variables in the 'x_B' assume non-negative integer values, the current solution is the optimal solution to the given linear integer programming problem. But, if some of the basic variables do not have non-negative integer value, an additional linear constraint called the Gomory constraint is generated. After having generated a linear constraint, it is added to the bottom of the optimal table so that the solution no longer remains feasible. The new problem is then solved by using the dual simplex method. If the optimal solution so obtained is again non-integer, another cutting plane is generated. The procedure is repeated until all basic variables assume non-negative integer values.

The procedure of developing a cut is discussed below. In the optimal simplex table, select one of the rows, called source row for which basic variable is non-integer. The desired cut is developed by considering only fractional parts of the coefficients in the source row. For this reason, such a cut is also referred to as fractional cut.

Suppose the basic variable x_r has the largest fractional value among all basic variables restricted to be integers. Then

the r^{th} constraint equation from the simplex table can be rewritten as:

$$x_{Br}(= b_r) = 1.x_r + (a_{r1}x_1 + a_{r2}x_2 + ...)$$
$$= x_r + \sum_{j \neq r} a_{rj}x_j \qquad \ldots (2.2.1.1)$$

where $x_j (j = 1, 2, 3,)$ represents all the non-basic variables in the r^{th} constraint except the variables x_r and $b_r (= x_{Br})$ is the non-integer value of variable x_r. Let us decompose the coefficients of x_j, x_r variables and x_{Br} into integer and non-negative fractional parts in Eqn. (2.2.1.1) as shown below:

$$[x_{Br}] + f_r = (1 + 0)x_r + \sum_{j \neq r} \{[a_{rj}] + f_{rj}\}x_j \qquad \ldots (2.2.1.2)$$

where $[x_{Br}]$ and $[a_{rj}]$ denote the largest integer obtained by truncating the fractional part from x_{Br} and a_{rj} respectively.

Rearranging Eqn. (2.2.1.2) so that all the integer coefficients appear on the left-hand side, it gets

$$f_r\{[x_{Br}] - x_r - \sum_{j \neq r} [a_{rj}]x_j\} = \sum_{j \neq r} f_{rj}x_j \qquad \ldots (2.2.1.3)$$

where f_r is a strictly positive fraction $(0 < f_r < 1)$ while f_{rj} is a non-negative fraction $(0 \leq f_{rj} \leq 1)$.

Since all the variables are required to assume integer values, terms in the bracket on the left hand side as well as on the right hand side must be non-negative numbers. Since the left-hand side in Eqn. (2.2.1.3) is f_r plus a non-negative number, it may written in the form of the following inequalities:

$$f_r \leq \sum_{j \neq r} f_{rj} x_j \qquad \ldots\ldots (2.2.1.4)$$

ie, $$\sum_{j \neq r} f_{rj} x_j = f_r + s_g$$

ie, $$-f_r = s_g - \sum_{j \neq r} f_{rj} x_j \qquad \ldots\ldots (2.2.1.5)$$

where s_g is a non-negative slack variable and is called the Gomory slack variable.

Equation (2.2.1.5) represents Gomory's cutting plane constraint. When this new constraint is added to the bottom of optimal simplex table, it would create an additional row to the table along with a column for the new variable s_g.

2.2.2 Steps of Gomory's All Integer Programming Algorithm

An iterative procedure for the solution of an all integer programming problem by Gomory's cutting plane method may be summarized in the following steps.

Step 1 Initialization

Formulate the standard linear integer programming problem. If there are any non-integer coefficients in the constraint equations, convert them into integer coefficients. Solve it by simplex method, ignoring the integer requirement of variables.

Step 2 Test the optimality

a) Examine the optimal solution. If all basic variables have in-

teger values, the integer optimal solution has been derived and the procedure should be terminated. The current optimal solution obtained in Step 1 is the optimal basic feasible solution to the linear integer programming.

b) If one or more basic variables with integer requirements have non-integer solution values, then go to Step 3.

Step 3 Generate cutting plane

Choose a row r corresponding to a variable x_r which has the largest fractional value f_r and generate the cutting plane as explained earlier in Eqn. (2.2.1.5).

$$-f_r = s_g - \sum_{j \neq r} f_{rj} x_j$$

where $0 \leq f_{rj} < 1$ and $0 < f_r < 1$

If there are more than one variable with the same largest fraction, then choose one that has the smallest contribution to the maximization linear programming problem or the largest cost to the minimization linear programming problem.

Step 4 Obtain the new solution

Add the cutting plane generated in Step 3 to the bottom of the optimal simplex table as obtained in Step 3. Find a new optimal solution by using the dual simplex method, i.e. choose a variable to enter into the new solution having the smallest ratio: $\{(c_j - z_j)/y_{ij}; y_{ij} < 0\}$ and return to Step 2. The process

is repeated until all basic variables with integer requirements assume non-negative integer values.

2.3 Gomory's Mixed-Integer Programming problem [Ham95] [Sha04]

It is important to note that the fractional cut assumes that all the variables, including slack and surplus, are integers. This means that the application of the fractional cut will yield no feasible integer solution unless all variables assume integer values. Simultaneously, for example, consider the constraint

$$\tfrac{1}{2}\ x_1 + x_2 \leq \tfrac{11}{3}$$

or

$$\tfrac{1}{2}\ x_1 + x_2 + s_1 = \tfrac{11}{3}$$

$$x_1, x_2, s_1 \geq 0 \text{ and integers}$$

It may be noted that this equation can have a feasible integer solution in x_1 and x_2 only if s_1 is non-integer. This situation can be avoided in two ways.

1. The non-integer coefficients in the constraint equation can be removed by multiplying both sides of the constraint with a proper constant. For example, the given constraint above is multiplied by 6 to obtain: $3x_1 + 6x_2 = 22$. However, this type of conversion is possible only when magnitudes of the integer coefficients are small.

2. Use a special cut called Gomory's mixed-integer cut or simply mixed-cut where only a subset of variables may assume integer values and the remaining variables remain continuous.

2.3.1 For Constructing Additional Constraint (Cut)

Consider the following mixed integer programming problem:

$Maximize\ Z = c_1x_1 + c_2x_2 + ... + c_nx_n$

Subject to the linear constraints

$a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1$

$a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n = b_2$

. .

. .

. .

$a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n = b_m$

and x_j are integers; j=1,2,...,k ($k < n$).

Suppose that the basic variable x_r is restricted to be integer and has a largest fractional value among all those basic variables which are restricted to take integer values. Then rewrite the r^{th} constraint (row) from the optimal simplex table as follows

$$x_{Br} = x_r + \sum_{j \neq r} a_{rj}x_j \qquad \ldots\ldots (2.3.1.1)$$

where x_j represents all the non-basic variables in the r^{th} row except variable x_r and x_{Br} is the non-integer value of variable x_r.

Decompose coefficients of x_j, x_r variables and x_{Br} into integer and non-negative fractional parts as shown below:

$$x_{Br} = [x_{Br}] + f_r$$

and $R_+ = \{j : a_{rj} \geq 0\}$ set of subscripts j (columns in simplex table for which $a_{rj} \geq 0$

$R_- = \{j : a_{rj} < 0\}$, set of subscripts j (column in simplex table for which $a_{rj} < 0$

Then Eqn(2.3.1.1) can be rewritten as:

$$[x_{Br}] + f_r = (1+0)x_r + \sum_{j \in R_-} a_{rj}x_j + \sum_{j \in R_+} a_{rj}x_j \quad \ldots (2.3.1.2)$$

Rearrange the terms in Eqn(2.3.1.2) so that all of the integer coefficients appear on the right-hand side.

$$\sum_{j \in R_+} a_{rj}x_j + \sum_{j \in R_-} a_{rj}x_j = f_r + \{\lfloor x_{Br} \rfloor - x_r\} = f_r + 1 \quad \ldots (2.3.1.3)$$

where f_r is a strictly positive fraction number (i.e. $0 < f_r < 1$), and I is the integer value.

Since the terms in the bracket on the right-hand side of Eqn. (2.3.1.3) are integers, left-hand side in Eqn (2.3.1.3) is either positive or negative according as $f_r + 1$ is positive or

negative.

Case 1 Let $f_r + 1$ be positive. Then it must be $f_r, 1 + f_r, 2 + f_r, ...$and then shall have

$$\sum_{j \in R_+} a_{rj} x_j + \sum_{j \in R_-} a_{rj} x_j \geq f_r \qquad \ldots\ldots (2.3.1.4)$$

Since $a_{rj} \in R_-$ are non-positive and $x_j \geq 0$.

$$\sum_{j \in R_+} a_{rj} x_j \geq \sum_{j \in R_+} a_{rj} x_j + \sum_{j \in R_-} a_{rj} x_j$$

and hence

$$\sum_{j \in R_+} a_{rj} x_j \geq f_r$$

Case 2 Let $f_r + 1$ be negative. Then it must be $f_r, -1 + f_r, -2 + f_r, ...$ and shall have

$$\sum_{j \in R_-} a_{rj} x_j \leq \sum_{j \in R_+} a_{rj} x_j + \sum_{j \in R_-} a_{rj} x_j \leq -1 + f_r \ldots\ldots (2.3.1.5)$$

multiplying both sides of Eqn.(2.3.1.5) by the negative number$(\frac{f_r}{f_r - 1})$,

$$(\frac{f_r}{f_r - 1}) \sum_{j \in R_-} a_{rj} x_j \geq f_r \qquad \ldots\ldots (2.3.1.6)$$

Either of inequalities (2.3.1.4) and (2.3.1.6) holds, since in both the cases the left-hand side is non-negative or one of these is greater than or equal to f_r. Thus, any feasible solution to mixed-integer programming must satisfy the following inequality:

$$\sum_{j \in R_+} a_{rj} x_j + (\frac{f_r}{f_r - 1}) \sum_{j \in R_-} a_{rj} x_j \geq f_r) \qquad \ldots\ldots (2.3.1.7)$$

Inequality (2.3.1.7) is not satisfied by the optimal solution

of the linear programming problem without integer requirement. This is because by putting $x_j = 0$ for all j, the left-hand side becomes zero, and right-hand side becomes positive. Thus, inequality (2.3.1.7) defines a cut.

Adding a non-negative slack variable s_g the Eqn. (2.3.1.7) can be rewritten as

$$s_g = -f_r + \sum_{j \in R_+} a_{rj} x_j + (\tfrac{f_r}{f_r - 1}) \sum_{j \in R_-} a_{rj} x_j \qquad \ldots\ldots (2.3.1.8)$$

Equation (2.3.1.8) represents the required Gomory's cut.

For generating the cut (2.3.1.8) it was assumed that the basic variable x_r should take integer value. But if one or more x_j, $j \in R_-$ are restricted to be integers, it can proceed as follows to improve a cut shown as Eqn. (2.3.1.7) to be a better cut, i.e. the coefficients $a_{rj}(\frac{f_r}{f_r - 1}), j \in R_-$ are desired to be as small as possible. The value of the coefficients of x_r can be increased or decreased by an integral amount in Eqn(2.3.1.3) so as to get a term with smallest coefficients in Eqn(2.3.1.7). Because in order to reduce the feasible region as much as possible through cutting planes, the coefficients of integer variable x_r must be as small as possible. The smallest positive coefficient for x_r in Eqn(2.3.1.3) is :

$\{f_{rj}; \frac{f_r}{1 - f_r}(1 - f_{rj})\}$

The smaller of the two coefficients would be considered to make the cut (2.3.1.8) penetrate deeper into the original

feasible region. A cut is said to be deep if the intercepts of the hyper plane represented by a cut with the x-axis are larger. Obviously,

$$f_r \leq \frac{f_r}{1-f_{rj}}(1-f_{rj}); f_{rj} \leq f_r$$

and $$f_r > \frac{f_r}{1-f_{rj}}(1-f_{rj}); f_{rj} > f_r$$

Thus, the new cut can be expressed as

$$s_g = -f_r + \sum_{j \in R} f^*_{rj} x_j \qquad \ldots\ldots (2.3.1.9)$$

where

$$f^*_{rj} = \left\{ \begin{array}{c} a_{rj};\ a_{rj} \geq 0\ and\ x_j\ non-integer \\ (\frac{f_r}{f_r-1})a_{rj};\ a_{rj} < 0\ and\ x_j\ non-integer \\ f_{rj};\ f_{rj} \leq f_r\ and\ x_j\ non-integer \\ (\frac{f_r}{f_r-1})a_{rj};\ f_{rj} > f_r\ and\ x_j\ non-integer \end{array} \right\} (2.3.1.10)$$

2.3.2 Steps of Mixed-Integer Programming Algorithm

Mixed-integer cutting plane method can be summarized in the following steps:

Step 1 Initialization

Formulate the standard linear integer programming problem. Solve it by simplex method ignoring integer requirement of variables.

Step 2 Test of optimality

1. Examine the optimal solution. If all integers restricted basic variables have integer values, then terminate the procedure. The current optimal solution obtained in Step 1 is the optimal basic feasible solution to the linear integer programming problem.

2. If all the integer restricted basic variables are not integers, then go to Step 3.

Step 3 Generate cutting plane

Choose a row r corresponding to a basic variable x_r which has largest fractional value f_r and generate a cutting plane as explained earlier in the form [2.3.1.10]

$$s_g = -f_r + \sum_{j \in R_+} a_{rj} x_j + \left(\frac{f_r}{f_r - 1}\right) \sum_{j \in R_-} a_{rj} x_j$$

$$\text{where } 0 < f_r < 1$$

Step 4 Obtain the new solution

Add the cutting plane generated in Step 3 to the bottom of the optimal simplex table as obtained in Step 1. Find a new optimal solution by using dual simplex method and return to Step 2. The process is repeated until all restricted basic variables are integers.

2.4 Branch and Bound Algorithm [Ham04] [Kan00] [Sha04]

The branch and bound method first divides the feasible region into smaller subsets, and then examines each of them successively until a feasible solution that gives optimal value of objective function is obtained. This technique is applicable to both the Integer Programming Problem, pure as well as mixed and involves the continuous version of the problem.

This method involves a well-structured systematic search of the space of all feasible solutions of constrained optimization problems. Usually the space of all feasible solutions is repeatedly portion into smaller and smaller subsets(Branching) and a lower bound is calculated for the cost of solutions with in each subset (Bound). After each partitioning, those subsets with a bound that exceeds the cost of a known feasible solution are excused from all further partitioning. Thus large subsets of solutions may be excluded from consideration without examining each solution in these subsets. The partitioning continues until a feasible solution is found such that its cost is no greater than the bound for any subset.

The success of this technique depends upon the number of solutions examined before an optimal solution is reached. Branch and Bound algorithms have been used with some degree of success to solve integer programming problems, the scheduling problem, the traveling salesman problem, the plant

location problem, the assignment problem and a variety of other problems with a finite number of feasible solutions.

2.4.1 Branching

A first approximation to the solution of any integer program may be obtained by ignoring the integer requirement and solving the resulting linear program by one of the techniques already presented. If the first approximation contains a variable that is not integral. Say X_j^*, then $i_1 < X_j < i_2$, where i_1 and i_2 are consecutive, non-negative integers. Two new integer program with either the constraint $X_j \geq i_1$ or the constraint $X_j \leq i_2$. This process called Branching, has the effect of shrinking the feasible region in a way that eliminates from further consideration the current non-integral solution for X_j but still preserves all possible solutions to original problem.

For the two integer programs created by the branching process, first approximations are obtained by again ignoring the integer requirements and solving the resulting linear programs. If either first approximation is still non integral, then the integer program which gave rise to that first approximation becomes a candidate for further branching.

2.4.2 Bounding

Assume that the objective function is to be Maximized. Branching continues until an integral first approximation is

obtained (which is thus an integral solution).

The value of the objective for this integral solution becomes a lower bound for the problem, and all programs whose first approximations, integral or not, yield values of the objective function smaller than the lower bound are discarded. Branching continues from those programs having non-integral first approximations that give values of the objective function greater than the lower bound. If in the process, a new integral solution is uncovered having a value of the objective function greater than the current lower bound, then this value of the objective function becomes the new lower bound. The program that yielded the old lower bound is eliminated, as all programs whose first approximations give values of the objective function smaller than the new lower bound. The branching process continues until there are no programs with non-integral first approximations remaining under consideration. At this point the current lower bound solution is the optimal solution to the original integer program.

If the objective function is to be minimized, the procedure remains the same, except that upper bounds are used. Thus the value of the first integral solution becomes an upper bound for the problem, and the programs eliminated when their approximate Z-values are greater than the current upper bound.

One always branches from that program which appears most nearly optimal. When there are a number of candidates for further branching, one chooses that having the largest Z-value, if the objective function is to be Maximized, or that having the smallest Z-value, if the objective function is to be minimized.

2.4.3 Step by Step procedure

The branch and bound method first divides the feasible region into smaller subsets, and then examines each of them successively unit a feasible solution that gives optimal value of objective function is obtained.

The iterative procedure is summarized below.

Step 1 Obtain an optimum solution of the given Linear Programming Problem ignoring the restriction of integers.

Step 2 Test the integer ability of the optimum solution obtained in Step 1.

There are two cases

Step 2.1 If the solution is in integers then the current solution is optimum to the given integer programming problem.

Step 2.2 If the solution is not in integers then goto next step.

Step 3 Considering the value of objective function as upper bound obtain the lower bound by rounding off to integral values of the decision variables.

Step 4 Let the optimum value X_j^* of the variables X_j is not in integers. Then subdivide the given Linear Programming Problem into two problems

Sub-problem 1: Given Linear Programming Problem with an additional constraint $X_j \leq [X_j^*]$

Sub-problem 2 : Given Linear Programming Problem with an additional constraint $X_j \geq [X_j^*] + 1$

where $[X_j^*]$ is the largest integer constraint in X_j^*.

Step 5 Solve the two sub-problems obtained in Step 4. There may arise three cases.

Step 5.1 If the optimum solutions of the two sub-problems are integral, then the required solution is one that gives larger value of Z.

Step 5.2 If the optimum solution of one sub-problem is integral and the other sub-problem has no feasible optimum solution, the required solution is same as that of the sub-problem having integer valued solution.

Step 5.3 If the optimum solution of one sub-problem is integral while that of the other is not integer valued

then record the integer valued solution and repeat Step 3 and 4 for the non integer valued sub-problem.

Step 6 Repeat Step 3 to 5, until all integer valued solutions are recorded.

Step 7 Choose the solution amongst the recorded integer valued solutions that yields an optimum value of Z.

2.5 Balas Additive Algorithm[Ham04][Kan00[Sha04]

The zero-one problem must start dual-feasible, that is, optimal but not feasible. Moreover, all the constraints must be of the type($\leq$), thus ruling out explicit equations. The additive algorithm is now presented by using the concept of partial solutions. The exclusion tests used to fathom partial solutions and augment new variables at level one are also generalized for the zero-one problem.

Consider the following general binary problem.

Minimize $Z = \sum_{i=1}^{n} c_j x_j, \qquad$ all $c_j \geq 0$

Subject to

$\sum a_{ij} X_j + S_i = b_i, \quad$ i=1,2,...,m

$X_j = 0$ or $1, \quad$ for all j

$S_i \geq 0, \qquad$ for all i

Let J_t be the partial solution at node t (initially, $J_0 \equiv$

$\emptyset$, which means that all variables are free) and assume z^t is the associated value of z while $\bar{z}$ is the current best upper bound(initially $\bar{z}$=8).

Test 1 For any free variables x_r, if $a_{ir} = 0$ for all i corresponding to $S_i^t < 0$, then x_r cannot improve the feasibility of the problem and must be discarded as non-promising.

Test 2 For any free variable x_r, if

$c_r + z^t = \bar{z}$ then x_r cannot lead to an improved solution and hence must be discarded.

Test 3 Consider the i^{th} constraint

$a_{i1}x_1 + a_{i2}x_2 + ... + a_{in}x_n + S_i = b_i$ for which $S_i^t < 0$. Let N_t define the set of free variables not discarded by tests 1 and 2. None of the free variables in N_t are promising if for at least one $S_i^t < 0$, the following condition is satisfied.

$$\sum_{j \in N_t} min\{0, a_{ij}\} > S_i^t$$

This actually says that the set N_t cannot lead to a feasible solution and hence must be discarded altogether. In this case, J_t is said to be fathomed.

Test 4 If $N_t \neq \emptyset$, the branching variable x_k is selected as the one corresponding to

$$v_k^t = \max_{j \in N_t} \{v_j^t\}$$

where

$$v_k^t = \sum_{i=1}^{m} \min\{0, S_i^t - a_{ij}\}$$

If $v_k^t = 0, x_k = 1$ together with J_t yields an improved feasible solution. In this case, J_{t+1}, which is defined by J_t with k augmented on the right, is fathomed. Otherwise, the foregoing tests are applied again to J_{i+1} until the enumeration is completed, that is, until all the elements of the fathomed partial solution are negative.

2.6 Branch and Cut [Whol98] [Nem88]

The bounds obtained from the Linear Problem relaxations are often weak, which may causes standard Branch and Bound algorithm to fail in practice. It is therefore of crucial importance to tighten the formulation of the problem to be solved. The idea of dynamically adding so called cutting planes to the problem is one way of obtaining stronger bounds. Combining the cutting plane algorithm with Branch and Bound results in the very powerful class of Branch and Cut algorithms.

In order to find integral solution to an Integer Problem, the cutting plane algorithm can be hybridized with Branch and Bound, resulting in the Branch and Cut algorithm. The idea is to generate cutting planes throughout the Branch and Bound tree of a standard Branch and bound algorithm, in order to get tight bounds at each node. In practice, many issues such as cut pool management, different cut generation

strategies, Branch and Bound strategies and many more have to be considered.

2.6.1 Step by Step procedure

The step by step procedure of the Branch and Cut algorithm is given below:

Step 1 Initialization

Denote the initial integer programming problem by ILP^0 and set the active nodes to be L = $\{ILP^0\}$. Set the upper bound to be$Z = +\infty$. Set $Z_1 = -\infty$ for the one problem $l \in L$.

Step 2 Termination

If $L = \phi$,then the solution x^* which yielded the incumbent objective value Z is optimal. If no such x^* exists (i.e., Z=$+\infty$) then ILP is infeasible.

Step 3 Problem selection

Select and delete a problem ILP^1 from L.

Step 4 Relaxation

Solve the linear programming relaxation of ILP^1. If the relaxation is infeasible, set $Z = +\infty$ and go to Step 6. Let Z_1 denote the optimal objective value of the relaxation if it is finite and let x^{lR} be an optimal solution; otherwise set $Z_1 = -\infty$.

Step 5 Add cutting planes

If desired, search for cutting planes that are violated by x^{lR}; if any are found, add them to the relaxation and return to Step 4.

Step 6 Fathoming and Pruning

Step 6.1 If $Z_1 \geq Z$ go to Step 2.

Step 6.2 If $Z_1 < Z$ and x^{lR} is integral feasible, update Z= Z_1, delete from L all problems with Z_1 = Z, and go to step 2.

Step 7 Partitioning

Let $\{S^{1j}\}_{j=1}^{j=k}$ be a partition of the constraint set S^1 of problem ILP^l. Add problems$\{S^{1j}\}_{j=1}^{j=k}$ to L, where ILP^{lj} is ILP^l with feasible region restricted to S^{lj} and Z_{lj} for j=1, 2,...,k is set to the value of Z_1 for the parent problem *l*. Go to step 2.

2.7 Branch and Price [Bar98] [Wol98]

In Branch and Price, the concept of column generation is combined with the Branch and Bound algorithm.

2.7.1 Column Generation

The simplex algorithm is at the origin of the column generation concept, where only variables with negative reduced costs are allowed to enter the basis in each iteration. Given a Linear Problem with a large number of variables, it would be

efficient to consider only the variables potentially improving the objective function. The main idea is to efficiently determine a variable with negative reduced costs to enter the basis, add it to the problem, resolve it and iteratively repeat this process until no variable with negative reduced costs exists anymore.

2.7.2 Branch and Price

Since column generation is an algorithm for solving Linear Problems, it has to be combined with another method in order to solve Integer Problems to optimality. The so called Branch and Price algorithm [Bar98] is the result of combining column generation with Branch and Bound. In each node of the Branch and Bound tree, column generation is performed to solve the Linear Problem relaxation. Branching is usually performed on original variables or by other strategies to partition the remaining search space in a balanced way.

An important point is that the column generation algorithm used has to be aware of branching decisions and may only generate solutions respecting them. Another interesting question is whether the column generation algorithm should search for optimal solutions of the pricing problem or not. From a theoretical point of view, Branch-and-Cut and Branch-and-Price are closely related, since column generation in the primal problem corresponds to cut generation in the dual and

vice versa.

2.8 Conclusion

Methods used to find the optimal solution to a Linear Integer Programming Problem were discussed. In earlier the cutting plane method, the optimum integer solution will be reached definitely after introducing enough new constraints to eliminate all the superior non-integer solution. In this method one or more new constraints (Gomory's constraints) added for integral solution. After adding the Gomory's constraints the problem is solved by dual simplex method to get an optimal integral solution. Hence the construction of such constraints is very important and tedious job and needs special attentions. This method leads more operations and time considerations.

In Branch and Bound method space of all feasible solutions is repeatedly portioned into smaller and smaller subsets names Branching and a lower bound is calculated for the cost of solutions with in each subset called Bound. After each partitioning those subsets with a bound that exceeds the cost of a known feasible solution are excused from all further partitioning. In this method the partitioning and calculation of Bounding solution is repeatedly done. So this method is also takes large number of multiply/divide operation and long time.

Both Branch-and-Cut and Branch-and-Price is not spe-

cialized in solving Integer Programming Problems and because many "expensive" matrix operations are required when solving the Linear Problem relaxation. In the next chapter the proposed Linear Integer Programming Problem method is to be presented.