Column Generation

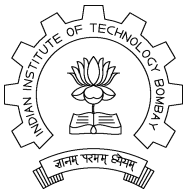
MTech Seminar Report

by

Soumitra Pal
Roll No: 05305015

under the guidance of

Prof. A. G. Ranade
Computer Science and Engineering
IIT-Bombay



Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai

Acknowledgments

I would like to thank my guide, Prof. A. G. Ranade, for his consistent motivation and directions throughout this work.

Soumitra Pal
MTech-I
CSE, IIT Bombay

Abstract

Column Generation is a technique for solving (mixed) integer programming problems with larger number of variables or columns. This technique was first applied to large real life cutting stock problem by Gilmore and Gomory. Since then several researchers have applied the column generation technique to many real life applications.

In this report, we present basics of the theory of column generation for general linear programs. The quality of the solution of integer programs using LP relaxation depends on the problem formulation. We discuss three formulations of the cutting stock problem (or related bin packing problem) and compare the bounds of them. The column generation based formulations give better bounds. We outline a generic algorithm based on *column generation* and *branch-and-bound*, commonly known as *branch-and-price*, to solve large scale integer programs. We discuss a flow-formulation based implementation of this algorithm.

Contents

Chapter 1

Introduction

The successful solution of large scale mixed integer programming (MIP) problems requires solving linear programming (LP) relaxations that have huge number of variables. One of several reasons for considering LP with huge number of variables is that compact formulation of a MIP may have a weak LP relaxation. Frequently the relaxation can be tighten by using huge number of variables.

In the pricing step of the revised simplex algorithm, if the optimal solution is not already reached, the algorithm figures out the non-basic variable that should replace one of the basic variables to improve the objective function in the next iteration of the algorithm. This explicit search of variable, may become intractable for problems with huge number of variables (or columns). Gilmore and Gomory [6], [7] proposed an ingenious way of solving this problem. The algorithm starts with a few columns and generates new column as and when required by solving a subproblem or oracle.

Gilmore and Gomory applied this column generation technique to solve large scale cutting stock problem. Since then several researchers have applied the column generation technique to many real life applications.

We introduce the column generation technique using an example of the cutting stock problem. In section 1.1 we describe the problem. We gradually build the context for the column generation technique. In section 1.2 we show a simple model for solving the problem. Linear programming solution of the model does not give proper answer and some time fails. We give a better model in section 1.3. We discuss different issues in solving the model and how column generation technique can address them in section 1.4

1.1 Cutting stock problem

The cutting stock problem arises from many physical applications in industry. For example, in a paper mill, there are a number of rolls of paper of fixed width waiting to be cut (these rolls are called *raws*), yet different manufacturers want different numbers of rolls of various-sized widths (these rolls are called *finals*). How should the rolls be cut so that the least amount of left-overs are wasted? or rather, least number of rolls are cut? This turns

out to be an optimization problem, or more specifically, an integer linear programming problem.

Example An instance of the problem. The raw rolls all have width of $W = 10$ inches. There are orders for $m = 4$ different finals of widths as follows:

i	Quantity Ordered b_i	Order Width w_i (inches)
1	9	3
2	79	5
3	90	6
4	27	9

1.2 Kantorovich model

For solving the problem, the first thing we need is a formulation of problem as an integer program.

One model can be formed as follows. Suppose, there are K raws. We keep K binary variables y_k's, $k \epsilon \{1..K\}$. y_k is assigned to 1 if k^{th} raw is cut at all; to 0 otherwise. We also keep $m \times K$ integer variables x_{ik}'s, $i \epsilon \{1..M\}$. x_ik is assigned to the number of times finals of width w_i is cut in roll k.

Minimize

$$z = \sum_{k=1}^{K} y_k \tag{1.1}$$

subject to

$$\sum_{k=1}^{K} x_{ik} \geq b_i, i = 1..m \tag{1.2}$$

$$\sum_{i=1}^{m} w_i x_{ik} \leq W y_k, k = 1..K \tag{1.3}$$

$$y_k = 0 \text{ or } 1, k = 1..K \tag{1.4}$$

$$x_{ik} \text{ integer and } \geq 0, i = 1..m, k = 1..K \tag{1.5}$$

This formulation was introduced by Kantorovich. For solving the given example, we set $K = 9 + 79 + 90 + 27 = 215$ assuming at most one final is cut from one raw. We can set

value of K even greater than 215. In that case the all the extra variables y_k become 0 in the optimal solution.

Let us now think about the solution of the above integer program. One of the most obvious way is to relax the integrality constraints on the variables x_{ik} and y_k. In that case, the problem becomes a linear program which can be solved using standard algorithms such as the simplex algorithm.

For the given example, solution to the LP relaxation is 120.5 which is significantly less that the integer solution 157. In other examples having more finals, the gap increases. Hence just rounding the values of the variables in the optimum solution of the LP relaxation will be far from the integral solution. Also, corresponding to the optimum LP solution, the value 120.5 is assigned to one of the y_ks. The rest are all assigned to zero. If we add the constraint

$$y_k \leq 1, k = 1..K \tag{1.6}$$

the value 120.5 gets distributed as 120 of the y_ks having value 1 and one more y_k having 0.5. Rest become all 0. This is definitely not the solution we want.

1.3 Gilmore-Gomory model

The solution improves significantly if we use the following trickier formulation introduced by Gilmore and Gomory. The possible cutting patterns are enumerated beforehand. The patterns are described by the vector $(a_{1j}, \cdots, a_{ij}, \cdots, a_{mj})$ where element a_{ij} represents the number of rolls of width w_i obtained in cutting pattern j. Let x_j be a decision variable that designates the number of rolls to be cut according to cutting pattern j.

minimize

$$z = \sum_{j \epsilon J} x_j \tag{1.7}$$

subject to

$$\sum_{j \epsilon J} a_{ij} x_j \geq b_i, i = 1, 2, \cdots m \tag{1.8}$$

$$x_j \text{ integer and } \geq 0, \forall j \epsilon J \tag{1.9}$$

where J is the set of valid cutting patterns.

For the given example, the valid cutting patterns are

$$
\begin{array}{llll}
1 & 10 & \geq & 9 \\
2 & 10 & \geq & 6 \\
3 & 10 & \geq & 6+3 \\
4 & 10 & \geq & 5 \\
5 & 10 & \geq & 5+5 \\
6 & 10 & \geq & 5+3 \\
7 & 10 & \geq & 3 \\
8 & 10 & \geq & 3+3 \\
9 & 10 & \geq & 3+3+3
\end{array}
$$

Similar to the Kantorovich formulation, this can also be solved using the LP relaxation. However, values of x_js (we say x_j^*) may not be all integer in the optimal solution of the LP relaxation. In that case, we can round each x_j^* down to the nearest integer and obtain a solution close to actual integer solution. The residual demands of finals which are not met due to rounding down can be found by brute-force.

For the given example, optimal LP solution $z = 156.7$ corresponds to

$x_1^* = 27$, $x_3^* = 90$, $x_5^* = 39.5$

Rounding each x_j^* down to nearest integer gives value of $z = 157$ which fortunately matches with the optimal integer solution.

1.4 Column generation approach

However, there are difficulties which originate from two different reasons:

- Problems commonly encountered in the paper industry may involve huge number of variables. For example, if the raw rolls are 200 in. wide and if the finals are ordered in 40 different lengths ranging from 20 in. to 80 in., then the number of different patterns may easily exceed 10 or even 100 million. In that case, the solution may not be tractable.
- Passing from an optimal *fractional-valued* solution to an optimal *integer-valued* solution is not easy. Rounding the fractional values down and satisfying the residual demands, as we mentioned in the example, may not yield the optimal working plan. If the finals are ordered in small enough quantities, then the patterns used in the optimal integer valued solution may be quite different from those used originally in the optimal fractional valued solutions.

An ingenious way of getting around the first difficulty was suggested by P. C. Gilmore and R. E. Gomory. The trick is to work with only a few patterns at a time and to generate new patterns only when they are really needed. This technique is called *delayed column generation*. We discuss this in details in the chapter 2.

No efficient way of handling the second difficulty is known. However there have been instances of tacking this difficulty by using a method commonly known as *branch-and-price*. Here the trick is to combine the delayed column generation with the standard

branch-and-bound algorithm for solving integer programs. Delayed column generation is applied to each node of the branch and bound tree to solve the LP subproblem. This is discussed in details in the chapter 4.

By now, the reader should have an idea about the column generation technique to solve linear programs, particularly, huge integer programming problems. The rest of this report is organized as follows.

In chapter 2, we provide the mathematical foundation of column generation. We elaborate the rationale of column generation by providing an algebraic interpretation of the pricing step in simplex algorithm.

In the previous sections, we have mentioned about two formulations of the cutting stock problem: *Kantorovich formulation* which is straight forward but having poor LP relaxation value and *Gilmore-Gomory formulation* which is trickier but gives better LP relaxation. It is amenable to column generation.

In chapter 3, we present some concepts from the Dantzig-Wolfe decomposition, and indicate how Gilmore-Gomory model can be obtained applying a Dantzig -Wolfe decomposition to Kantorovich model. We also argue why the second model gives better LP relaxation bounds. We also provide flow based formulation of the problem.

In chapter 4, we present the outline of the generic branch-and-price algorithm. We also discuss an exact solution of the flow-based model using an implementation of the branch-and-price algorithm.

We conclude the report in chapter 5.

Chapter 2

Mathematical Foundation of Column Generation

In this chapter, we touch upon the mathematical theory required to understand the basics of column generation. We provide details up-to the level that is required to understand the column generation technique.

We start in section 2.1 with the algebraic interpretation of the simplex algorithm, particularly the step in which a non-basic variable replaces a basic variable to improve the cost. We show how this leads to process of delayed column generation in section 2.2.

2.1 Pricing step of simplex method

Here we see how the pricing step in the revised simplex method works. In this step one non-basic variable replaces one basic variable so that the value of the cost function improves. For more details refer Chvatal [2].

Let us call the following linear program the *master problem* (MP). Here it is convenient to consider the primal linear program with equality constraints.

$$Z_{LP} := \max cx \tag{2.1}$$

such that

$$Ax = b, x \in R_+^n. \tag{2.2}$$

Its dual is

$$W_{LP} := \min ub \tag{2.3}$$

such that

$$uA \geq c, u \in R^m. \tag{2.4}$$

We suppose that rank$(A) = m \leq n$, so that all the redundant equations have been removed from the LP.

For solving the LP, in each iteration of the simplex method we look for a non-basic variable to price out and enter the basis. Column generation plays a key role in this pricing step. We explain the algebra of pricing step below.

Let $A = (a_1, a_2, \cdots, a_n)$ where a_j is the jth column of A. Since rank$(A) = m$, there exists an $m \times m$ nonsingular submatrix $A_B = (a_{B_1}, a_{B_2}, \cdots, a_{B_m})$. Let $J = \{1, 2, \cdots, n\}$ and $B = \{B_1, B_2, \cdots, B_m\}$ and let $N = J \setminus B$. Now permute the columns of A so that $A = (A_B, A_N)$. We can write $Ax = b$ as $A_B x_B + A_N x_N = b$, where $x = (x_B, x_N)$. Then a solution to $Ax = b$ is given by $x_B = A_B^{-1} b$ and $x_N = 0$.

$$Ax = \begin{bmatrix} A_B & A_N \end{bmatrix} \begin{bmatrix} x_B \\ 0 \end{bmatrix} = b, \text{ or } x_B = A_B^{-1} b. \tag{2.5}$$

If x is a genuine corner (it is feasible) provided $x_B \geq 0$. Its cost is

$$cx = \begin{bmatrix} c_B & c_N \end{bmatrix} \begin{bmatrix} x_B \\ 0 \end{bmatrix} = c_B A_B^{-1} b. \tag{2.6}$$

The question is where to go next, after leaving this corner. The decision is made easy by elimination on A, which reduces the square part B to the identity matrix. In matrix notation this multiplies $Ax = b$ by A_B^{-1}

$$\begin{bmatrix} I & A_B^{-1} A_N \end{bmatrix} \begin{bmatrix} x_B \\ 0 \end{bmatrix} = A_B^{-1} b. \tag{2.7}$$

Now suppose, the non-basic (zero) components of x is increased to some non-zero value. Say x is changed to $x' = [x'_B x'_N]$. To maintain equality 2.7,

$$\begin{bmatrix} I & A_B^{-1} A_N \end{bmatrix} \begin{bmatrix} x'_B \\ x'_N \end{bmatrix} = A_B^{-1} b, \tag{2.8}$$

or

$$x'_B + A_B^{-1} A_N x'_N = A_B^{-1} b, \tag{2.9}$$

or

$$x'_B = A_B^{-1} b - A_B^{-1} A_N x'_N \tag{2.10}$$

That changes cost to

$$cx' = c_B x'_B + c_N x'_N = c_B (A_B^{-1} b - A_B^{-1} A_N x'_N) + c_N x'_N \tag{2.11}$$

Rearranging this into

$$cx^{'} = c_B A_B^{-1} b + (c_N - c_B A_B^{-1} A_N) x_N^{'} \tag{2.12}$$

The reduction in cost $cx^{'} - cx$ is

$$c_B A_B^{-1} b + (c_N - c_B A_B^{-1} A_N) x_N^{'} - c_B A_B^{-1} b = (c_N - c_B A_B^{-1} A_N) x_N^{'} \tag{2.13}$$

We can see that as x_N increases, the cost function goes up or down depending on the sign of the vector $\bar{c}$ in parentheses.

$$\bar{c} = c_N - c_B A_B^{-1} A_N \tag{2.14}$$

This vector contains the reduced costs. If $\bar{c} \geq 0$ then the current corner is optimal. The product $\bar{c} x_N$ in equation 2.13 cannot be negative since $x \geq 0$, so the best decision is to keep $x_N = 0$ and stop. On the other hand, suppose a component is negative. Then the cost is reduced if the corresponding non-basic variable enters into the basis. The simplex method chooses one entering variable - generally the one with the most negative component of $\bar{c}$.

2.2 Delayed Column Generation

In each iteration of the simplex method we look for a non-basic variable to price out and enter the basis. That is, in the pricing step, given the dual vector $u = c_B A_B^{-1}$ we need to find the non-basic variable corresponding to which the reduced cost component is negative. One way to find this out is to calculate the minimum of all components of the reduced cost vector and take if it is negative.

$$\arg\min \{\bar{c}_j = c_j - u a_j | j \in N\} \tag{2.15}$$

where a_j is a column of the matrix A.

If the columns in A_B are also considered in calculating reduced cost vector, corresponding components in the reduced cost becomes 0 since $c_B - c_B A_B^{-1} A_B = 0$. Thus, inclusion of the basic columns do not change the decision. The equation 2.15 can be re-written as

$$\arg\min \{\bar{c}_j = c_j - u a_j | j \in J\} \tag{2.16}$$

An explicit search of j from J may be computationally impossible when $|J|$ is huge. Gilmore and Gomory showed that the searching j from J explicitly is not necessary. If we look carefully, it is not j that we are interested. Rather we are interested in the column a_j that can replace one column of the basic matrix A_B. In practical applications, these columns often represent combinatorial objects such as paths, patterns, sets, permutations. They follow some embedded constrains i.e. $a_j \in \mathcal{A}$, where $\mathcal{A}$ represents the constrained set. For example, in the cutting stock problem with Gilmore-Gomory formulation, each

column represents a cutting pattern, which must satisfy the knapsack constraint - sum of widths of finals in a particular cutting pattern must not exceed the width of the raw rolls.

Thus, in practice, one works with a reasonably small subset $J^{'} \subset J$ of columns, with a *restricted master problem* (RMP). Assuming that we have a feasible solution, let $\bar{\lambda}$ and $\bar{u}$ be primal and dual optimal solutions of the RMP, respectively. When columns a_j, $j \in J$, are given as elements of a set $\mathcal{A}$ and the cost coefficient c_j can be computed from a_j, $c_j = c(a)$, then the subproblem or oracle

$$\bar{c}^* = \min \{c(a) - \bar{u}a | a \in \mathcal{A}\} \tag{2.17}$$

gives answer to the pricing problem.

For the cutting stock problem, $c_j = 1, \forall j \in J$ and the set $\mathcal{A}$ is given by the constraint

$$\sum_{i=1}^{m} a_{ij} w_i \leq W, \forall j \in J \tag{2.18}$$

Thus, for the cutting stock problem, the subproblem is

$$\max \sum_{i=1}^{m} \bar{u}_i a_i \tag{2.19}$$

$$\text{such that } \sum_{i=1}^{m} a_i w_i \leq W \tag{2.20}$$

$$a_i \geq 0 \tag{2.21}$$

This way of starting with a basic set of columns & generating more columns as and when necessary is known as *Delayed Column Generation* or simply *Column Generation*.

With the usual precautions against cycling of simplex method, column generation is finite and exact. In addition, it is possible to utilize the knowledge of intermediate solution quality during the process. Let $\bar{z}$ denote the optimal objective function value to the RMP. Note by duality we have $\bar{z} = \bar{u}b$. Interestingly, when an upper bound $\kappa \geq \sum_{j \in J} \lambda_j$ holds for an optimal solution of the master problem, we establish not only an upper bound on z^* in each iteration, but also a lower bound. We can not reduce $\bar{z}$ by more than κ times the smallest reduced cost $\bar{c}^*$, hence

$$\bar{z} + \bar{c}^* \kappa \leq z^* \leq \bar{z} \tag{2.22}$$

In the optimum solution of the MP, $\bar{c}^* = 0$ for the basic variables and the bounds close. When the objective already is a sum of variables that is $c = 1$, we use z^* instead of κ and obtain the improved lower bound $\bar{z}/(1 - \bar{c}^*) \leq z^*$. For $c \geq 0$, Farley [5] proposes a more general lower bound at the expense of a slightly increased computation effort. Let $j^{'} \in \arg\min_{j \in J} \{c_j / \bar{u}a_j | ua_j > 0\}$. Then

$$\bar{z} + c_{j'}/\bar{u}a_j \leq z^* \leq \bar{z} \tag{2.23}$$

This knowledge about the running bound helps in deciding whether to continue generating columns at a particular node in the branch-and-bound tree. We discuss branch-and-bound with column generation in chapter 4.

Chapter 3

Different Formulations and Bounds

In chapter 1 we have shown two different formulations - Kantorovich and Gilmore-Gomory formulations - for the cutting stock problem. In this chapter we discuss the two models with more details - quality of the two models and how one can be derived from other using the Dantzig-Wolfe decomposition. Carvalho [4] provides a fairly detailed description of these two and other models for cutting stock and related bin packing problems.

In section 3.1 we discuss about the LP bounds of the Kantorovich model. In section 3.2 we introduce the Dantzig-Wolfe decomposition. We show how Gilmore-Gomory model can be obtained from Kantorovich model by applying the Dantzig-Wolfe decomposition in section 3.3.

In section 3.4 we show another formulation. The formulation uses the concept of flow in an acyclic directed graph.

3.1 Kantorovich model

For the ease of reference, we restate the Kantorovich formulation

Minimize

$$z = \sum_{k=1}^{K} y_k \tag{3.1}$$

subject to

$$\sum_{k=1}^{K} x_{ik} \geq b_i, i = 1..m \tag{3.2}$$

$$\sum_{i=1}^{m} w_i x_{ik} \leq W y_k, k = 1..K \tag{3.3}$$

$$y_k = 0 \text{ or } 1, k = 1..K \tag{3.4}$$

$$x_{ik} \text{ integer and } \geq 0, i = 1..m, k = 1..K \tag{3.5}$$

where y_k is assigned to 1 if k^{th} raw is cut at all; to 0 otherwise. x_{ik} is assigned to the number of times finals of width w_i is cut in roll k.

A lower bound for the optimum can be obtained from the solution of the LP relaxation, which results from substituting the two last constraints for $0 \leq y_i \leq 1$ and $x_{ij} \geq 0$. This bound can be very poor i.e. the difference of this solution from the actual integer solution is significant. Martello e Toth showed that this bound is equal to $\lceil \sum_{i=1}^{m} \frac{w_i b_i}{W} \rceil$. This bound is equal to minimum amount of space that is necessary to accommodate all the finals if there is a way to reuse the wastes by pasting them together. Hence, the bound is very poor in instances with large waste. In the limit, as W increases, when all the items have a size $\lfloor W/2 + 1 \rfloor$, the bound approaches to 1/2 of the integer solution.

For the example cutting stock problem given in the chapter 1, the bound is 120.5 which is equal to $(3 * 9 + 5 * 79 + 6 * 90 + 9 * 27)/10$. The integer solution is 156. This gap increases, if the number of orders for width $6 \sim \lfloor W/2 + 1 \rfloor$ increases.

3.2 Dantzig-Wolfe decomposition

The Dantzig-Wolfe decomposition is a powerful tool that can be used to obtain stronger linear programming formulations for integer and combinatorial optimization problems.

Many integer programming models have nice structure and a constraint set that can be partitioned and expressed as follows:

$$\min cx \tag{3.6}$$

$$\text{subject to } Ax = b \tag{3.7}$$

$$x \in X \tag{3.8}$$

$$x \geq 0 \text{ and integer} \tag{3.9}$$

where the constraint set given by equation 3.8 have nice structure.

The LP relaxation of this model that results from dropping the integrality constraints on the variable x can be very weak. A stronger model can be obtained by restricting the set of points from set X that are considered in the LP relaxation of the reformulated model.

According to Minkowski's theorem, any point of an non-empty polyhedron X can be expressed as a convex combination of the extreme points of X, denotes as x^1, x^2, $\cdots$,

$x^{|P|}$, plus a non-negative linear combination of the extreme rays of X, denoted as r^1, r^2, $\cdots$, $r^{|R|}$:

$$X = \{x \in R_+^n : x = \sum_{p \in P} \lambda_p x^p + \sum_{r \in R} \mu_r r^r, \sum_{p \in P} \lambda_p = 1, \lambda_p \geq 0, \forall p \in P, \mu_r \geq 0, \forall r \in R\} \tag{3.10}$$

where $P = x^p$ is the set of extreme points of X and $R = r^r$ is the set of extreme rays of X.

Substituting the value of x in the original model, we obtain the following model

$$\min\ c(\sum_{p \in P} \lambda_p x^p + \sum_{r \in R} \mu_r r^r) \tag{3.11}$$

$$\text{subject to } A(\sum_{p \in P} \lambda_p x^p + \sum_{r \in R} \mu_r r^r) = b \tag{3.12}$$

$$\sum_{p \in P} \lambda_p = 1 \tag{3.13}$$

$$\lambda_p \geq 0, \forall p \in P \tag{3.14}$$

$$\mu_r \geq 0, \forall r \in R \tag{3.15}$$

After rearrangements of the terms, we have:

$$\min\ \sum_{p \in P} (cx^p)\lambda_p + \sum_{r \in R} (cr^r)\mu_r \tag{3.16}$$

$$\text{subject to } \sum_{p \in P} (Ax^p)\lambda_p + \sum_{r \in R} (Ar^r)\mu_r = b \tag{3.17}$$

$$\sum_{p \in P} \lambda_p = 1, \lambda_p \geq 0, \forall p \in P, \mu_r \geq 0, \forall r \in R \tag{3.18}$$

In Dantzig-Wolfe decomposition, this problem is usually denoted as master problem. Its decision variables are λ_p, $\forall p \in P$, and μ_r, $\forall r \in R$, and its columns corresponds to the extreme points and extreme rays of X.

In the formulation obtained using Dantzig-Wolfe decomposition has huge number of variables and usually solved by column generation. Usually the subproblem that generates columns for the reformulated model from the set X is a well structured optimization problem and solved using standard techniques.

When all the extreme points and extreme rays of X are integers i.e. the subproblem has integrality property, the LP relaxation of the original model and relaxation of the

reformulated model have the same LP bounds. However, when the subproblem does not have integrality property the relaxation of the reformulated model provides better LP bound.

This happens in the cutting stock problem as we will see in the next section.

3.3 Gilmore-Gomory model

We show that the Gilmore-Gomory model can be obtained from the Kantorovich model by applying the Dantzig-Wolfe decomposition. The decomposition is applied by keeping constraints 3.2 in the master problem and using knapsack constraints 3.3 for defining the subproblem.

Kantorovich model has several knapsack constraints, one for each roll k. Each knapsack constraint defines a bounded set, having only extreme points, and no extreme rays. The set X is the intersection of the knapsack constants for each roll. The solution set of the knapsack constraint of roll k can be restricted by eliminating valid fractional solutions of the knapsack constraints that do not correspond to valid cutting patterns for roll k.

Only integer solutions to the knapsack constraint for roll k are considered, i.e. those such that $\sum_{i=1}^{m} w_i a_{ik} \leq W, a_{ik} \geq 0$ and integer. These solutions are described by the vector $(a_{1k}^p, \cdots, a_{ik}^p, \cdots, a_{mk}^p)^T, \forall p \in P$, and the reformulated model can be obtained is as follows:

If $y_k = 0$ then by constraints 3.3, $x_{ik} = 0, \forall i$. Otherwise, by Minkowski's theorem $x_{ik} = \sum_{p\in P} \lambda_k^p a_{ik}^p, \sum_{p\in P} \lambda_k^p = 1$. Combining the two cases, we can write $x_{ik} = \sum_{p\in P} \lambda_k^p a_{ik}^p, \sum_{p\in P} \lambda_k^p \leq 1$ i.e. excluding $x_{ik} = 0$. We can also see that when $y_k = 1$, y_k can be replaced by $\sum_{p\in P} \lambda_k^p$. When $y_k = 0$, there is only one point in the knapsack polytope $x_{ik} = 0$ and hence $\lambda_k = 0$. So irrespective of value of y_k, it can be replaced by $\sum_{p\in P} \lambda_k^p$. Substituting x_{ik} and y_{ik} in Kantorovich model, we get,

$$\min \sum_{k=1}^{K} \sum_{p\in P} \lambda_k^p \tag{3.19}$$

$$\text{subject to } \sum_{k=1}^{K} \sum_{p\in P} a_{ik}^p \lambda_k^p \geq b_i, \forall i \tag{3.20}$$

$$\sum_{p\in P} \lambda_k^p \leq 1, k = 1, 2, \cdots, K \tag{3.21}$$

$$\sum_{p\in P} a_{ik}^p \lambda_k^p \geq 0, \text{ and integer }, \forall i, k = 1, 2, \cdots, K \tag{3.22}$$

$$\lambda_k^p \geq 0, \forall p \in P, k = 1, 2, \cdots, K \tag{3.23}$$

It can be seen that the LP relaxation of this model has only those feasible solutions that are non-negative linear combination of the integer solutions to the knapsack problem. Non-negative linear combinations of the fractional extreme points of the knapsack polytope, which are feasible in the linear relaxation of the Kantorovich model are eliminated.

Due to this fact, the reformulated model gives better bound of the LP relaxation.

The subproblem decomposes into K subproblems, which are knapsack problems. When all the rolls have the same width, all the subproblems are identical, and the reformulated model is equivalent to the Gilmore-Gomory model. If the K rolls used are grouped according to the cutting pattern that are cut to, the model takes of the form of Gilmore-Gomory model

$$\min \sum_{j \in J} x_j \tag{3.24}$$

$$\text{subject to } \sum_{j \in J} a_{ij} x_j \geq b_i, \forall i \tag{3.25}$$

$$x_j \geq 0, \forall J \in J. \tag{3.26}$$

where a_{ij} represents the number of rolls of width w_j obtained in cutting pattern j. x_j is the decision variable denoting the number of rolls to be cut according to pattern j.

It has been found that in real life cutting stock problems the following conjecture to be true.

Proposition 3.3.1 (Conjecture) *The gap between the optimal value of the cutting stock problem in the form of Gilmore-Gomory model and its LP relaxation is less than 2.*

This emphasizes the importance of this model.

3.4 Flow model

In this section we describe a flow-formulation for Bin-packing problem which is very similar to cutting stock problem. See Carvalho [3] for more details.

The bin packing problem can be stated as follows. Given a positive integer number of bins of capacity W and a list of n items of integer sizes $L = \{w_1, w_2, ..., w_n\}$ $(0 \leq w_i \leq W)$, the problem is to assign the items to the bins so that the capacity of the bins is not exceeded and the number of bins used is minimized.

There are many similarities between the bin-packing problem and the one dimensional cutting stock problem. In the cutting stock problem, the items of equal size, that are usually ordered in large quantities, are grouped into orders with required levels of demand. The cutting stock problem can be thought of as a bin-packing problem where the number of bins of a particular size is more than $1 (= b_i)$.

Given bins of integer capacity W and a set of different item sizes $w_1, w_2, \cdots, w_n$ the problem of determining a valid solution to a single bin can be modeled as the problem of finding a path in the acyclic directed graph with W + 1 vertices.

Consider a graph $G = (V, A)$ with $V = \{0, 1, \cdots, W\}$ and $A = \{(i, j) : 0 \leq i < j \leq W$ and $j - i = W_d$ for every d $\leq m\}$, meaning that there exists a directed arc between two vertices if there is an item of the corresponding size. The number of variables is $O(mW)$.

Consider additional arcs between $(k, k+1), k = 0, 1, \cdots, W-1$ corresponding to unoccupied portion of the bin. These arcs are known as loss arcs.

There is a packing in a single bin iff there is a path between the vertices 0 and W. Then length of arcs that constitute the path define the item sizes to be packed.

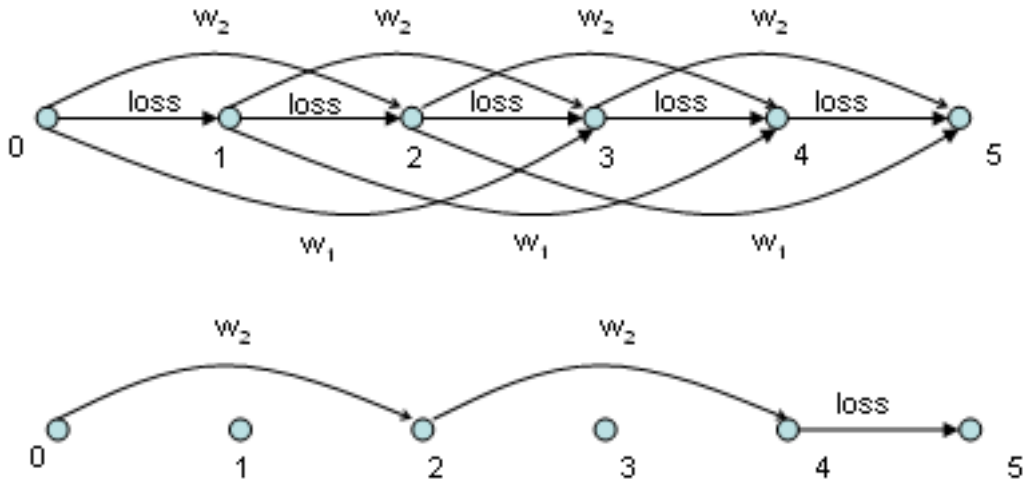


Figure 3.1: Flow graph and a cutting pattern

Figure 3.1 shows the graph associated to an instance with bin capacity $W = 5$ and items of size 2 and 3. In the same figure a path is also shown. The path corresponds to 1 items of size 2 and 1 item of size 3.

If a solution to a single bin corresponds to the flow of one unit between vertices 0 and W, a path carrying a larger flow will correspond to using the same packing solution in multiple bins. This leads us to the formulation of the problem.

The problem is formulated as the problem of determining the minimum flow between vertices 0 and W with additional constraints enforcing that the sum of the flows in the arcs of each order must be greater than or equal to the number of items of a given size.

Consider decision variables x_{ij} associated with the arcs defined above, which correspond to the number of items of size $j - i$ placed in any bin at a distance of i unit s from the beginning of the bin. The variable z can be seen as the total flow from vertex 0 to W.

The model is as follows :

Minimize

$$z \tag{3.27}$$

subject to

$$\sum_{(i,j)\in A} x_{ij} - \sum_{(j,k)\in A} x_{jk} = \begin{cases} -z, & \text{if } j = 0, \\ 0, & \text{if } j = 1, 2, \cdots W-1, \\ z, & \text{if } j = W; \end{cases} \tag{3.28}$$

$$\sum_{(k,k+w_d)\in A} x_{k(k+w_d)} \geq b_d, d = 1, 2, \cdots, m, \tag{3.29}$$

$$x_{ij} \text{ integer and } \geq 0, \forall (i,j) \in A \tag{3.30}$$

Carvalho [3] proves the following proposition.

Proposition 3.4.1 *The bounds of the linear programming relaxations of the flow model and Gilmore & Gomory model are equal.*

Using the variables presented thus far, there are many alternative solutions with exactly the same items in each bin. These alternative solutions, in other words, symmetry of the solution space cause the undesirable effect of same column getting generated repeatedly in the column generation process. Both the symmetry and size of the model can be reduced by selecting only a subset of arcs from A using the following criteria.

The arcs are specified such that the corresponding items are ordered in decreasing values of width.

Criterion 1 An arc of size w_e, designated by $x_{k,k_{w_e}}$, can only have its tail at a node k that is the head of another arc of size w_d, $x_{k-w_d,k}$, for $w_d \geq w_e$, or, else, from node 0, i.e., the left border of the bin.

In particular, if a bin has any loss, it will appear last in the bin. A bin can never start with a loss. Formally this can be stated as follows.

Criterion 2 All the loss arcs are $x_{k,k+1}$ can be set to zero for $k < w_m$.

In a bin, the number of consecutive arcs corresponding to a single item size must be less than or equal to the number of items of that size. Therefore, using criterion 1,

Criterion 3 Given any node k that is the head of another arc of size $w_d (w_d > w_e)$ or $k = 0$, the only valid arcs for size w_e are those that start at nodes $k + sw_e, s = 0, 1, 2, \cdots, b_e - 1$ and $k + sw_e \leq$, where b_e is the demand of items of size w_e.

Figure 3.2 shows the graph associated to an instance with bin capacity $W = 5$ and items of size 2 and 3. This is reduced as per the criteria 1 & 2.

Even after applying reduction using the above criteria, the model still can have symmetry. In instances with small average number of items per bin, which happens to be uncommon, if the criteria are applied there may have some symmetry. However, the symmetry is low and its undesirable effects are not so harmful.

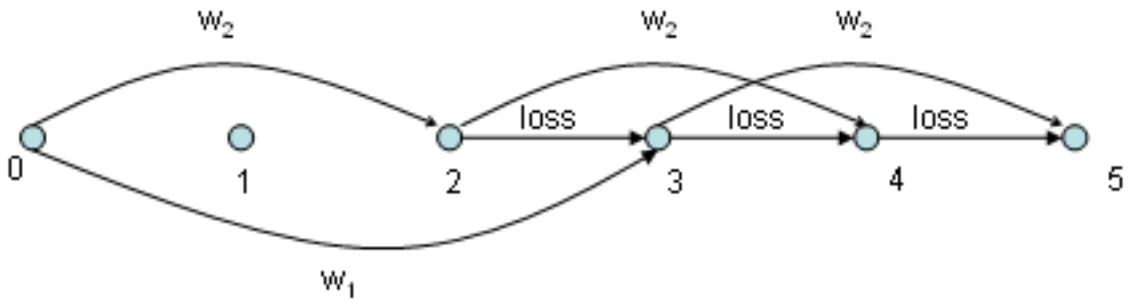


Figure 3.2: Reduced flow graph

Chapter 4

Branch-and-price algorithms

Even with the stronger LP relaxations, optimal solution to the LP relaxation may not give the optimal solution to the original problem. Among the standard approaches of solving integer programs using LP relaxations, the one we are interested uses column generation and branch-and-bound. Various notions have been coined for the synthesis of column generation and branch-and-bound such as *branch-and-price* Barnhart et al. [1] and *IP column generation* [8]. Besides the decomposition principles, essentially it all boils down to branch-and-bound.

In section 4.1 we outline the branch and price algorithm and the difficulties associated with it. In section 4.2 we discuss the implementation of this algorithm using the flow based formulation [3].

4.1 Generic Branch-and-price algorithm

Once we have a 'good' formulation for the IP problem the steps for solving using branch-and-price is straight forward. Barnhart et al. [1] describes the generic algorithm.

1. Start with an initial restricted master problem (RMP).
2. Solve the RMP using column generation. This may take multiple iterations before no more better column can be generated. The columns are generated by solving a subproblem.
3. If the LP solution to RMP is already feasible, stop. The solution is optimal.
4. Use branch-and-bound technique. Add new constraints to partition the solution space and branch. Apply the same technique of solving LP using column generation at each node of the search tree.

It may seem that branch-and-price involves nothing more than combining well known ideas for solving linear programs by column generation with branch-and-bound. However,

there are fundamental difficulties in applying column generation techniques for linear programming in IP solution methods. These include :

- Conventional integer programming branching on variables may not be effective because fixing variables can destroy the structure of the pricing problem.
- Solving these LPs and the subproblems to optimality may not be efficient, in which case different rules will apply for managing the branch-and-price tree.

Barnhart et al. [1] describes the different techniques of handling the above two issues. In the next section we see an implementation of the generic algorithm that gives exact solution to the bin-packing problem.

4.2 Branch and price algorithm for the flow model

Carvalho [3] provides a flow-formulation based solution to the cutting stock problem. In section 3.4 we have given the formulation. The algorithm follows the generic branch-and-price algorithm. In this section we discuss the special considerations taken by the algorithm.

4.2.1 Column generation subproblem

At each iteration, a subproblem is solved and a set of arcs (columns) is introduced in the restricted master problem. Let $u_j, j = 0, 1, \cdots, W$, be the dual variables associated with the flow conservations constraints 3.28 and $v_d, d = 1, 2, \cdots, m$, the dual variables associated with the demand constraints 3.29. Each column which corresponds to arc (i, j) has only three non-zero elements, i.e., a "-1" in row j and "+1" in the demand row d, corresponding to the ordered width $w_d = j - i$.

Let $\pi = (u, v)$ be the solution of the current basis of the restricted master problem. The reduced cost of a single column (arc) is equal to $\bar{c}_{ij} = -(-u_i + u_j + v_d)$.

As seen above, any solution can be expressed as a non-negative linear combination of paths. Therefore, instead of generating single arcs, sets of arcs that can be associated with a single path can be generated. Let $p = (0, a, b, \cdots, y, z, W)$ be a directed path starting at node 0 and ending at node W, and P the set of all valid paths.

To determine the most attractive path following subproblem is solved.

$$\text{minimize} \sum_{(i,j)\in A_{LP}} \bar{c}_{ij} x_{ij} \tag{4.1}$$

$$\text{subject to} \sum_{(i,j)\in A_{LP}} x_{ij} - \sum_{(j,k)\in A_{LP}} x_{jk} = \begin{cases} -1, & \text{if } j = 0, \\ 0, & \text{if } j = 1, 2, \cdots, W-1, \\ 1, & \text{if } j = W; \end{cases} \tag{4.2}$$

The arcs that correspond to the variables that are already in the restricted master problem are also considered in the subproblem. There is an attractive path if the optimum is strictly less than 0. Otherwise, the solution to the restricted master problem is optimal.

4.2.2 Branch-and-bound

The procedure used here is oriented to a problem where the gaps are almost always strictly smaller than one and can be reduced to zero by introducing the following valid inequality.

$\sum_{(k,k+1)\in A_{LP}} x_{k,k+1} \geq L_{min}$ where the minimum loss L_{min} is given by $\lceil z_{LP} \rceil W - \sum_{d=1}^{m} w_d b_d$.

The optimization problem is solved as a sequence of decision problems in which we want to find if there is, or is not, an integer solution with an objective value equal to the smaller known lower bound, which will always be an integer value, denoted as z_{LB}.

The first decision problem to be solved is to determine if there is an integer solution of value $\lceil z_{LP} \rceil$, i.e., $z_{LB} = \lceil z_{LP} \rceil$. As is known, most of the cutting stock problem instances have the integer round-up property, meaning that almost always the solution to this decision problem will provide an optimal integer solution to the optimization problem.

However, if an integer solution is not found in the search tree, meaning that the instance does not have integer roundup property, the lower bound has to be increased by one unit, i.e., $z_{LB} = z_{LB} + 1$, and the procedure has to be repeated. The application of this problem is justified in problems with very small gap between the solution to LP relaxation & actual integer solution.

Branching constraints are imposed on single variables of the master problem, starting with variables that correspond to larger widths and placed nearer the left border of the roll, i.e., the fractional variable with the smaller value i, and to break ties with the one with larger $j - i$. A depth-first search is performed, using branching constraints of the following type:

$x_{ij} \geq \lceil x_{ij} \rceil$ and $x_{ij} \leq \lfloor x_{ij} \rfloor$

In each node w of the search tree, we want to determine if there is any solution with objective value equal to z_{LB}. If not the node is fathomed. This can only be done when the generation procedure fails to produce more attractive columns and the value of the objective function of the restricted master problem z^w, is strictly larger than z_{LB}.

After a branching constraint is added, the restricted master problem is re-optimized, and one of the following cases occurs:

- the solution is integer, with the value equal to z_{LB}, which means that is optimal;
- the solution is fractional, with a value equal to z_{LB}, making it necessary to introduce new branching constraints;
- the solution has a value that is strictly greater than z_{LB}. The column generation procedure is called, trying to reach a solution with value z_{LB} leading either to case 1 or 2. If this is not possible, the node is fathomed.

The number of decision problems to be solved is finite, as there is an absolute performance ratio for the bin packing problem. Also the set of constraints that can be imposed in each decision problem is finite, which guarantees the finiteness of the entire solution process.

Figure 4.1 shows a flowchart for the column generation/branch-and-bound procedure.

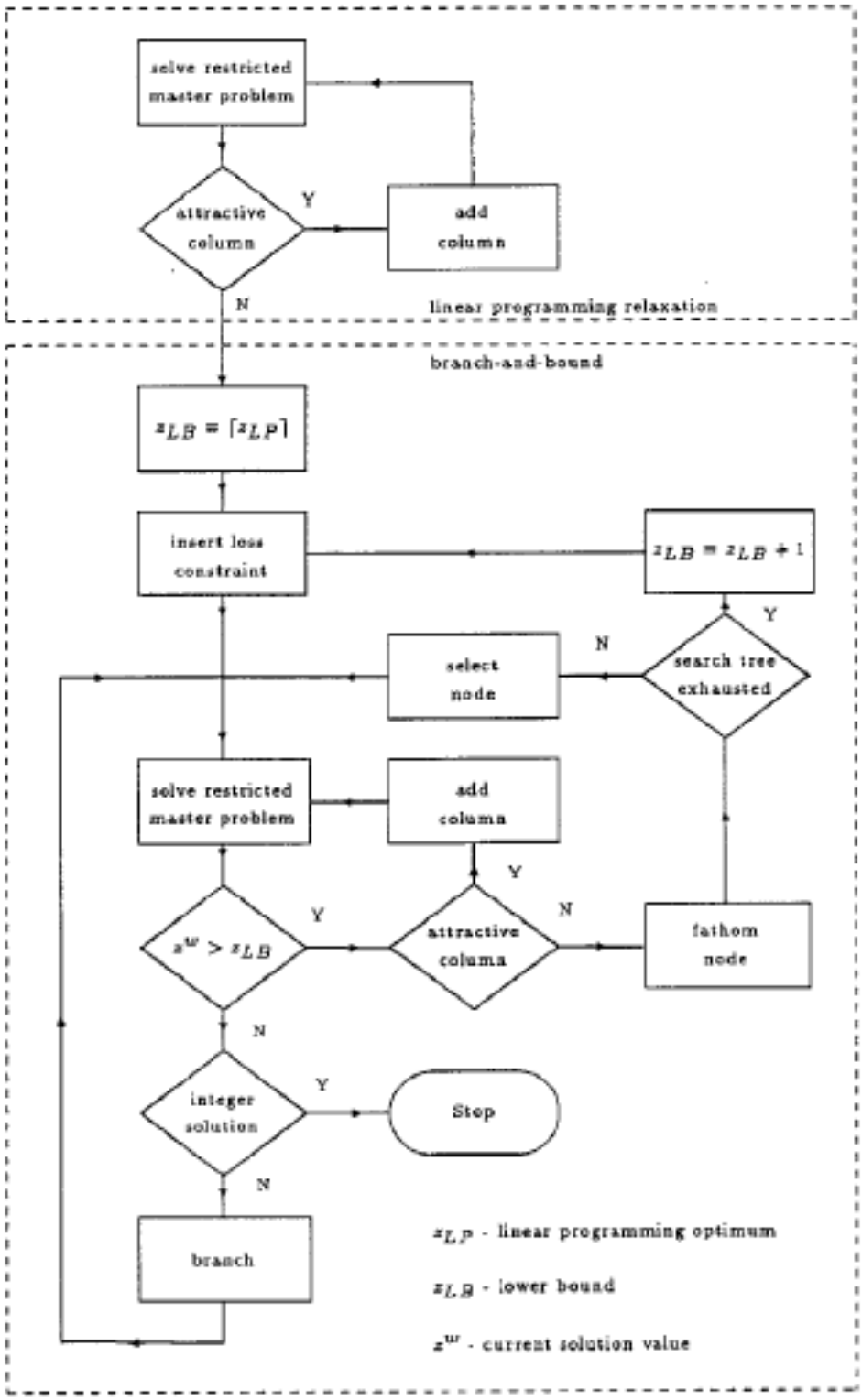


Figure 4.1: Flow chart for the algorithm

Chapter 5

Conclusions

Column generation is a success story in solving large scale integer programming. The advantage of keeping only few columns at any time, makes many real life integer optimization problems thought of being solved. When used with extensive formulation having stronger linear programming relaxation, the method gives quicker result. Application specific heuristics applied at each node of branch-and-bound tree also improves the efficiency of the procedure.

References

[1] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsberg, and P. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 48(3):318–326, 2000.

[2] V. Chvátal. *Linear Programming*, chapter 13. W. H. Freeman and Company, New York, 1983.

[3] J. M. Valério de Carvalho. Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research*, 86:629–659, 1999.

[4] J. M. Valério de Carvalho. Lp models for bin-packing and cutting stock problems. *European Journal of Operations Research*, 141(2):253–273, 2002.

[5] A. A. Farley. A note on bounding a class of linear programming problems, including cutting stock problems. *Operations Research*, 38(5):922–923, 1990.

[6] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9:849–859, 1961.

[7] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem - part ii. *Operations Research*, 11:863–888, 1963.

[8] F. Vanderbeck and L. A. Wolsey. An exact algorithm for ip column generation. *Operations Research Letters*, 19:151–159, 1996.