Improving Branch-And-Price Algorithms For Solving One Dimensional Cutting Stock Problem

M. Tech. Dissertation

Submitted in partial fulfillment of the requirements
for the degree of

Master of Technology

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

Acknowledgements

I would like thank my guide **Prof. Abhiram Ranade** for his guidance throughout this work. Without his support this work would not have been done.

I would also like to thank my parents, brother and sisters for their support all the time.

I would also like to acknowledge the generous support provided by my department and my institute.

Soumitra Pal
Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

Abstract

Branch-and-price is a well established technique for solving large scale integer programming problems. This method combines the standard branch-and-bound framework of solving integer programming problems with Column Generation. In each node of the branch-and-bound tree, the bound is calculated by solving the LP relaxation. The LP relaxation is solved using Column Generation.

In this report, we discuss our project on improving the performance of branch-and-price based algorithms for solving the industrial one-dimensional cutting stock problem. In the early part our project, we studied several branch-and-price based algorithms that are available in the literature for solving the cutting stock problem. We write down our findings from the study in this report. In the later part of the project, we worked on a few ideas to improve the performance of the algorithms. We publish the results in the report. We conclude the report by giving some directions for future works.

Contents

List of Tables

Chapter 1

Introduction

Branch-and-price is a well established technique for solving large scale integer programming problems. This method, combines the standard branch-and-bound framework of solving integer programming problems with 'Column Generation'. In each node of the branch-and-bound tree, the bound is calculated by solving the LP relaxation. The LP relaxation is solved using Column Generation. In this report, we discuss our work on improving the performance of the branch-and-price based algorithms for solving the industrial one-dimensional cutting stock problem.

In this chapter, we introduce the cutting stock problem and give an overview of how the problem is solved using branch-and-price. In section 1.1, we describe the problem. We give an integer programming formulation for solving the problem in section 1.2. As it is generally done, the first attempt should be to solve the problem by solving its LP relaxation. The main issue in solving the LP relaxation is the huge number of columns in the formulation. In section 1.3, we discuss how this issue is taken care of using Column Generation. Converting the LP solution to an integer solution is another difficult challenge. This is solved by using branch-and-price method described in section 1.4. We give the outline of a generic branch-and-price algorithm and its subtasks in section 1.5. Branch-and-price can be combined with the cutting plane approach of solving integer programming problems. This is described briefly in section 1.6. By the end of that section, the reader may have a fair introduction to the problem. Before concluding the chapter we state a conjectured property of the Gilmore-Gomory formulation in section 1.7.

We conclude the chapter by providing an outline of the rest of the chapters in section 1.8. In subsection 1.8.1, we give an overview of the literature survey that we conducted in the initial phases of the project. We give a brief discussion of our contribution in subsection 1.8.2.

1.1 One Dimensional 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 (these rolls are called *stocks*), yet different manufacturers want different numbers of rolls of various smaller widths (these rolls are called *items*). 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. The problem can be formally stated as:

- **Input:**
  - Stock width, W
  - Set of item widths, $\{w_1, w_2, \ldots, w_m\}$
  - Set of demands, $\{b_1, b_2, \ldots, b_m\}$
- **Output:**
  - Minimum number of stocks, z, required to satisfy the demands
  - The cutting patterns, $\{A_1, A_2, \ldots, A_z\}$, such that each pattern $A_j = (a_{1j}, a_{2j}, \ldots, a_{mj})$ fits in a stock, i.e. $\sum_{i=1}^{m} a_{ij} w_i \leq W$ for all $j = 1, \ldots, z$. a_{ij} denotes the number of instances of item i that are obtained if the stock is cut in pattern j.

There are different well-known variants of the cutting stock problem, with different width stocks, stocks cut in two dimension etc. However, we consider only the stated form of the problem.

Example Let us consider an instance of the problem. The stocks have width $W = 10$ inches. There are orders for $m = 4$ different items as given in Table 1.1.

Table 1.1: An instance of the 1D cutting stock problem

i	Width of items w_i (inches)	Quantity ordered b_i
1	3	9
2	5	79
3	6	90
4	9	27

1.2 Gilmore-Gomory Formulation

The cutting stock problem is commonly solved using the following formulation introduced by Gilmore and Gomory [1961, 1963]. The possible valid 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 instances of item 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.

$$\min \quad z = \sum_{j \in J} x_j \tag{1.1}$$

$$\text{s.t.} \quad \sum_{j \in J} a_{ij} x_j \geq b_i, \qquad i = 1, 2, \cdots m \tag{1.2}$$

$$x_j \text{ integer and } \geq 0, \qquad \forall j \in J \tag{1.3}$$

where J is the set of valid cutting patterns. For the given example, the valid cutting patterns are shown in Table 1.2

Table 1.2: All possible valid cutting patterns for the example 1D CSP problem

j	Validity	a_{1j}	a_{2j}	a_{3j}	a_{4j}
1	$10 \geq 9$	1	0	0	0
2	$10 \geq 6$	0	1	0	0
3	$10 \geq 6+3$	0	1	0	1
4	$10 \geq 5$	0	0	1	0
5	$10 \geq 5+5$	0	0	2	0
6	$10 \geq 5+3$	0	0	1	1
7	$10 \geq 3$	0	0	0	1
8	$10 \geq 3+3$	0	0	0	2
9	$10 \geq 3+3+3$	0	0	0	3

As an initial attempt, the formulation is solved using the LP relaxation. For the given example, optimal LP solution, $z = 156.7$, corresponds to $x_1^* = 27$, $x_3^* = 90$, $x_5^* = 39.5$. This result is not useful because cutting a pattern fractionally does not make sense. However, solving the LP relaxation is important. Later, in section 1.5, we will see that we need to solve the LP relaxation in each node of the branch-and-bound tree to obtain the integer solution.

1.3 Column Generation

Solving the LP relaxation is also not easy. The difficulty originates due to the huge number of valid patterns (and hence huge number of variables) involved in the instances commonly encountered in the industry. For example, if the stock rolls are 200 in. wide and if the items 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.

An ingenious way of getting around this difficulty was suggested by Gilmore and Gomory [1961, 1963]. The trick is to work with only a few patterns at a time and to generate new patterns only when they are really needed. The rationale is that, in the final solution, most of the variables would be zero anyway. This process is called *delayed column generation*. Here, we give a brief description of the process.

We first 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. Let us consider the following LP,

$$Z_{LP} = \min c^T x \qquad \text{s.t.} \qquad Ax = b, \qquad x \in R^n_+.$$

and its dual

$$W_{LP} = \max b^T \pi \qquad \text{s.t.} \qquad A^T \pi \geq c, \qquad \pi \in R^m.$$

We assume that $\text{rank}(A) = m \leq n$, so that all the redundant equations have been removed from the LP. Let $A = (a_{*1}, a_{*2}, \cdots, a_{*n})$ where a_{*j} is the jth column of A. Since $\text{rank}(A) = m$, there exists an $m \times m$ nonsingular submatrix $A_B = (a_{B_1}, a_{B_2}, \cdots, a_{B_m})$. Let $J = \{1, 2, \cdots, n\}$, $B = \{B_1, B_2, \cdots, B_m\}$, and $N = J \setminus B$. 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 $x_N = 0$. If we move from one solution x to another x', reduction in cost is

$$\begin{aligned}
c^T x' - c^T x &= c_B^T x'_B + c_N^T x'_N - c_B^T x_B - c_N^T x_N \\
&= c_B^T x'_B + c_N^T x'_N - c_B^T A_B^{-1} b && \text{replacing values of } x_B \text{ and } x_N \\
&= c_B^T (A_B^{-1} b - A_B^{-1} A_N x'_N) + c_N^T x'_N - c_B^T A_B^{-1} b && x' \text{ is also a solution of } Ax = b \\
&= (c_N^T - c_B^T A_B^{-1} A_N) x'_N
\end{aligned}$$

We can see that, as x_N increases, the cost function goes up or down depending on the sign of the reduced cost vector $\bar{c} = (c_N^T - c_B^T A_B^{-1} A_N)^T$ in parentheses. In each iteration of the simplex method, we look for a non-basic variable corresponding to which the reduced cost component is negative and bring it into the basis. That is, given the dual vector $\pi = (c_B^T A_B^{-1})^T$ (because it feasible to the dual), we need to find the cost-improving non-

basic variable. One way to find this out is to calculate the minimum of all components of the reduced cost vector and take it if it is negative. Thus we need to find

$$\arg\min\{\bar{c}_j = c_j - \pi^T a_{*j} \,|\, j \in N\} \tag{1.4}$$

If the columns in A_B are also considered in calculating reduced cost vector, corresponding components in the reduced cost becomes 0. Thus, inclusion of the basic columns does not change the decision. The equation (1.4) can be re-written as

$$\arg\min\{\bar{c}_j = c_j - \pi^T a_{*j} \,|\, j \in J\} \tag{1.5}$$

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 items in a particular cutting pattern must not exceed the width of the stock rolls.

Thus, in practice, one works with a reasonably small subset $J' \subset J$ of columns. The problem with this smaller set of columns is called *restricted master problem* (RMP). Assuming that we have a feasible solution, let $\bar{x}$ and $\bar{\pi}$ 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}, $c_j = c(a_{*j})$, then the subproblem or oracle

$$\bar{c}^* = \min\{c(a) - \bar{\pi}^T a \,|\, a \in \mathcal{A}\} \tag{1.6}$$

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

$$\sum_{i=1}^{m} a_{ij} w_i \leq W, \ \forall j \in J \tag{1.7}$$

Thus, for the cutting stock problem, the subproblem is

$$\max \sum_{i=1}^{m} \bar{\pi}_i a_i \qquad \text{s.t.} \qquad \sum_{i=1}^{m} a_i w_i \leq W, \qquad a_i \text{ integer and } \geq 0 \tag{1.8}$$

The new column is added to the RMP and the process is continued. The process stops

when no more new column with negative reduced cost can be found. That implies, the optimum is reached.

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

1.4 Branch-and-Price

The 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 items which are not met due to rounding down can be found by brute-force. In our example CSP (see Table 1.1), rounding each x_j^* down to nearest integer gives value of $z = 157$ which fortunately matches with the optimal integer solution. There is no guarantee that this process will always give an optimal integer solution. If the items 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.

However, this difficulty is solved 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.

The idea of a branch-and-bound algorithm is as follows. If the LP solution of the problem is not integral, then split the solution space into multiple subspaces so that the fractional solution is eliminated from all the subspaces, recursively solve the problem on the subspaces and take the best out of them. Moreover, if there is an 'indication' that the problem on a subspace can never be better than the best solution found so far, then skip recursing on that subspace. The recursion takes the form of a tree. This tree is known as branch-and-bound tree.

In case of a minimization problem, the 'indication' can be obtained from the LP relaxation of the problem. The LP solution acts as a global lower bound on the integer solution on the current subspace in consideration. Thus, if the global lower bound is greater than the best integer solution found so far, we can safely discard current subspace.

1.5 A Generic Branch-and-Price Algorithm

Here, we give the outline of a generic branch-and-price algorithm for the cutting stock problem. Actual implementations differ in the details of the individual steps of this generic algorithm.

1. Solve the problem with a heuristic solution. The *incumbent* is set with this solution.

The incumbent represents the best solution found so far. Let this solution be the set of columns in the RMP.

2. Form a branch and bound tree with a single node in it representing the entire solution space. Mark this node undiscovered and unexamined. A node is 'discovered' means its lower bound is already calculated. The node is 'examined' only if it is processed completely.

3. Choose an unexamined node in the branch and bound tree. If no examined node exists go to 6. If the node is discovered, go to 5, else go to 4.

4. (Node is undiscovered)

   (a) Get a lower bound LB on the solution space represented by the node. It is done by solving the LP relaxation using column generation.

   (b) Find a feasible integer solution X for the current node. If X is less than *incumbent*, let $incumbent = X$. It can be noted that if $X = LB$, it is the optimal integer solution for the subtree under the current node. However, this may not be the final integer solution because there can be better integer solutions on other branches not under the current subtree. Hence, mark the node only as discovered. Go to step 3.

5. (Node is discovered)

   (a) Mark this node examined.

   (b) If the lower bound value LB at this node is $\geq incumbent$, there is no need to explore this subtree and hence go to step 3.

   (c) Otherwise, divide the solution space at the node into two sets such that the current fractional solution is eliminated from both, create two nodes for the two solution subspaces, add them to the branch and bound tree, and mark both undiscovered and unexamined. This step is known as branching. Go to step 3.

6. Stop, *incumbent* gives the optimal solution value.

This is a very rough outline of a branch-and-price algorithm. We discuss the subtasks in the following subsections. However, the algorithms available in literature differ in detailed implementation of the subtasks. We discuss about them in chapter 2.

1.5.1 Initialization

The steps 1 and 2 are the initialization steps. At each branch-and-bound node, an initial feasible LP solution is required to start up the column generation procedure. The step 1 sets up the feasible solution for the root node.

One simple solution is to start the column generation procedure using an "unit matrix", i.e., a matrix made of one column for each item with a single non-zero entry, $a_i = \lceil W/w_i \rceil$. This initial solution, however is generally not good in terms of number of master iterations and columns generated. So, one uses better heuristic solutions such as well know Fast Fit Decreasing (FFD), Best Fit Decreasing (BFD) etc for the Bin Packing problem. It can be noted that a bin packing problem is an instance of the cutting stock problem where the items are ordered in small quantities.

1.5.2 LP Solution Using Column Generation

In step 4a, the LP is solved using column generation. The standard column generation procedure described in section 1.3 is followed here.

However, one of the main problems of solving integer programming problems using branch-and-price is the *tailing-off effect* or slow convergence of the column generation process at each node. It takes many iterations without much improvement in cost, particularly, at the end to get the optimal LP solution. This is a waste if the solution thus obtained is not included in the final solution. Moreover, the main objective of solving the LP is to obtain a lower bound on the optimal integer solution on the subspace corresponding to the current node. It will be enough if this objective is met.

Fortunately, it is generally not necessary to solve the LP to optimality to get the lower bound. The idea is as follows. Let us first describe the problem of slow convergence formally. Let z^u_{IP} be the integer solution to the problem on the space corresponding to node u. Let z^u_{LP} be the corresponding LP solution. Then z^u_{LP} is a lower bound on z^u_{IP}. However, at an intermediate step of column generation, we have only the LP solution for the restricted master problem, $\bar{z}^u_{LP}$. Unfortunately, $\bar{z}^u_{LP}$ is not a lower bound on z^u_{LP}, rather it is an upper bound on z^u_{LP}. Thus we can not use it as a lower bound on z^u_{IP} until we have reached the end of column generation when $\bar{z}^u_{LP} = z^u_{LP}$.

Let us know describe how this problem is solved. The solution is based on the theorem in Farley [1990], which states that if $\bar{c}$ is the objective of the subproblem solved last, then $\bar{z}^u_{LP}/\bar{c} \leq z^u_{LP}$. Thus the value $\bar{z}^u_{LP}/\bar{c}$ can be taken as a lower bound on z^u_{IP}. The column generation is terminated when the value exceeds the current incumbent. However, the algorithms in the literature uses lower bounds even tighter than this.

1.5.3 Rounding Heuristic

In step 4b of the generic algorithm, a heuristic solution is obtained from the LP solution. A rounding heuristic is a procedure that attempts to find a "good" integer solution by "rounding" the current integer solution. The difference between this heuristic and the heuristic used at the initialization step is that this time we are more equipped to get the

solution as we have already solved the LP relaxation. The performance of the branch-and-price depends on the goodness of the rounding heuristic. If this heuristic solution is good, we can quickly fathom the nodes whose LP relaxations themselves are higher than this heuristic solution.

Here, the standard procedure consists of the following steps. First, round down the fractional solution. Then, reduce demands by subtracting the columns that are fixed in the rounding down procedure. Finally, solve the problem with residual demands using some heuristic or exact procedure. The residual demands are generally small and can be solved comparatively faster.

1.5.4 Branching

In step 5c of the generic algorithm, the branching is performed. The challenge in formulating a branching rule is to find a scheme of separation of the feasible solution space. The scheme is applied successively at each node of the branch and bound tree eliminating fractional solutions. The scheme should have the following properties:

- It should exclude the current fractional solution and validly partition the solution space of the problem. In addition, an effective branching scheme should partition the solution space equally.
- There should be a guarantee that a feasible integer solution will be found (or infeasibility proved) after a finite number of branches.
- The branching makes the subproblem complex. This is because the subproblem must ensure that the column invalidated by the branching rule is not regenerated. This, in turn, makes sure that column generation can still be used to solve the more restricted LP relaxation at the descendant nodes. A good branching scheme should keep the master and subproblem tractable.

1.5.5 Node Selection

In step 3, a node from the set of active nodes are selected. The performance of a branch-and-price algorithm also depends on the node selection strategy. The common practice are the following

- *Best first search (bfs):* A node is chosen with the weakest lower bound (promising best solution). The goal is to improve the global lower bound which is the minimum of local bounds of all unprocessed leaves. However, the problem is that if the bound improves slowly, the search tree grows considerably.

- *Depth first search (dfs):* This rule chooses one of the deepest nodes in the tree. The advantages of dfs are the small size of the search tree and fast re-optimization of the subproblem. Also feasible solutions are found very quickly. The main disadvantage is that the global lower bound stays untouched for a long time resulting in bad solution guarantees, which leads to long optimality proof. Another drawback is if a wrong branch is chosen around the top, considerable time is spent in exploring the subtree which goes waste. For IRUP 1D-CSP instances, the correct LP bound is known from the beginning so pure dfs are quite efficient.

- *Diving from a bfs node:* This is a hybrid approach. At the beginning some number of nodes are chosen according bfs and after that their children are processed in the dfs fashion.

1.5.6 Summary

An implementation of the generic branch-and-price algorithm should implement all the tasks described above. However, if we have to explore different implementation of the same algorithm, we should look for the following

1. How does the algorithm implements branching? How are the branching constraints derived? How is the corresponding subproblem modified? How is the subproblem solved?

2. On what criteria, the algorithm should pick one unexamined node when there are several candidates?

3. How are the heuristic solutions obtained from the current LP solution? This also includes the heuristic used to obtain the initial solution.

4. How does it solve the slow convergence of the column generation?

In chapter 2, when we study the algorithms available in literature, we see how these issues are taken care of by them.

1.6 Branch-Cut-Price

One of the alternative approaches of getting integer solution from the LP solution is to add constraints (called cuts) one by one to the LP relaxation to eliminate fractional solutions until an integer solution is found. This type of algorithms are known as cutting plane algorithms. It is also possible to combine branch-and-price with cutting plane methods. The combined approach is known as branch-cut-price. In this method, in addition to the

standard branch-and-price procedures, the LP relaxation in each node are made tighter by adding cuts.

By this time, the reader may have a reasonably fair introduction of the problem we are going to solve, the standard branch-and-price algorithm for solving it and some higher level details of its subtasks. However, before concluding this introduction, we state a property of the Gilmore-Gomory formulation which will be used in the report.

1.7 Conjectured Property of Gilmore-Gomory Formulation

An instance of an integer (minimization) programming problem IP satisfies *Integer Round-Up Property (IRUP)* if the relation $Z_{IP} = \lceil Z_{LP} \rceil$ holds where Z_{IP} and Z_{LP} are the integer and LP optimal solutions respectively. Corresponding, the instance is also called an IRUP instance. If an instance does not satisfy IRUP, it called a Non-IRUP instance.

An instance of an integer (minimization) programming problem IP satisfies *Modified Integer Round-Up Property (MIRUP)* if the relation $Z_{IP} < \lceil Z_{LP} \rceil + 1$ holds. In other words, for an instance satisfying MIRUP, the integrality gap is strictly less than 2.

It is conjectured that the following proposition holds.

Conjecture 1.7.1 *All instances of the cutting stock problem with Gilmore-Gomory formulation satisfy MIRUP.*

With this background created, we give an overview of the rest of report.

1.8 Organization of the Report

1.8.1 Literature Survey

In chapter 2, we report our findings from the survey of a few branch-and-price algorithms and a branch-cut-price algorithm. Though the algorithms in literature implement the generic branch-and-price algorithm described in section 1.5, they differ in the detailed implementation of the subtasks. We study them and provide a summary.

1.8.2 Our Contribution

Though a considerable amount of time was spent in the literature survey, we were able come up with an implementation of the generic branch-and-bound algorithm and exper-

imented with a few alternative approaches to solve some of the subtasks of the generic branch-and-price algorithm. We tried the following ideas:

- accelerating the column generation procedure by modifying the master problem formulation
- solving the subproblems using a dynamic programming algorithm.
- obtaining a heuristic integer solution quickly from the LP solution of the master problem

We provide details of the implementation of these ideas in chapter 3 and present the experimental results in chapter 4.

We conclude the report in chapter 5. There, we also mention how our work can be extended in future.

Chapter 2

Literature Survey

In chapter 1, we described the generic branch-and-bound algorithm to solve the 1D cutting stock problem. We provided a brief description of the different subtasks of this generic algorithm. We also mentioned that the algorithms existing in the literature differ in the details of the implementation of the subtasks. In this chapter, we give the details of the algorithms we studied. The literature we covered are mainly Degraeve and Schrage [1999], Degraeve and Peeters [2003], Vance et al. [1994], Vance [1998], Vanderbeck [1999]. Instead of describing these publications one after another, we discuss them according to the subtasks of the generic algorithm. In each of the sections 2.1 through 2.5, we discuss one of the subtasks.

In section 2.1, we discuss the branching rules and the corresponding modification to the subproblems. We start with a simple scheme of branching on a single variable in subsection 2.1.1. A special branching scheme for the binary cutting stock problem is discussed in subsection 2.1.2. Two different implementations of the branching in general CSP are provided in subsections 2.1.4 and 2.1.3. Both of them use a set of variables for branching but differ in the choice of the variables.

In section 2.2, we discuss the solutions to the modified subproblems. Node selection is discussed in section 2.3. In section 2.4, we describe rounding heuristics. The commonly used rounding heuristics are mentioned in subsection 2.4.1. In subsection 2.4.2, we describe another rounding procedure named as Sequential Value Correction method. Section 2.5 describes the intermediate lower bounds used for early termination of column generation.

In section 2.6, we discuss a branch-cut-price algorithm provided in Belov and Scheithauer [2006]. We provide our conclusion from the survey in section 2.7. There, we mention the tasks that we identified to improve the performance of branch-and-price algorithms.

2.1 Branching Constraints and Modified Subproblem

We start with the two most important subtasks of the generic branch-and-price algorithm – (i) Devising branching constraints and (ii) Solving the modified subproblem. Since the form of the subproblem is determined by the branching rules, in this section, we mention the different branching decisions that are followed in literature and the corresponding subproblem formulations. The solutions to the subproblems are described in the section 2.2.

2.1.1 Conventional Branching on a Single Column

The obvious choice for branching is to branch on a single fractional variable. Degraeve and Schrage [1999], Degraeve and Peeters [2003], and Vance [1998] used this scheme.

Branching Scheme

Branching is done on a single fractional variable, say $x_j = \alpha$, α is fractional, by adding a simple upper bound $x_j \leq \lfloor \alpha \rfloor$ at the left branch and a simple lower bound $x_j \geq \lceil \alpha \rceil$ at the right branch.

Modified Subproblem

We need to take care of the effect of this branching in the corresponding subproblem. We discuss the effect on the two branches separately. The effect on the right branch is easier to handle. Since, in this case, x_j for the column j is $\geq \lceil \alpha \rceil$, adding the constraint is equivalent to reducing the demand vector by the column j multiplied by $\lceil \alpha \rceil$ and solving the residual problem. The residual problem can be solved as if it is a new problem. We do not need to modify the subproblem at all. If the same column j appears in the solution of the residual problem with a value β, we make $x_j = \lceil \alpha \rceil + \beta$ in the final solution.

The problem comes in the left branch where an upper bound on the decision variable is added. Since the variable is forcefully upper bounded, it is quite possible that the column corresponding to the variable will once again be generated by the subproblem. This is because, according to the new dual prices of the new restricted master, it may still be an optimal column. The subproblem should make sure that it does not regenerate this 'forbidden' column. As new nodes are created, more forbidden columns are added to master LP. We formalize the subproblem in a general node, as a Bounded Knapsack Problem with Forbidden Solutions (BKPFS). The details are given below.

Bounded Knapsack Problem with Forbidden Solutions (BKPFS)

Let us consider the problem where a knapsack of capacity W should be filled using m given item types, where type i has a profit π_i, weight w_i, and a bound b_i on the availability. A configuration is a set of items whose weight-sum does not exceed capacity. A configuration can be represented by $a = (a_1, a_2, \ldots, a_m),\ a_i \in \{0, 1, \ldots, b_i\},\ \sum_{i=1}^{m} w_i a_i \leq W$. We are also provided with a forbidden of set configurations $S = \{A_1, A_2, \ldots, A_n\}$. The problem is to find out a configuration such that the profit-sum of the included items is maximized and it is not in the forbidden set. The Bounded Knapsack Problem with Forbidden Solutions (BKPFS) may thus be defined as the following Integer Programming problem:

$$\max \quad \sum_{i=1}^{m} \pi_i a_i \tag{2.1}$$

$$\text{s.t.} \quad \sum_{i=1}^{m} w_i a_i \leq W \tag{2.2}$$

$$a_i \in \{0, 1, \ldots, b_i\} \qquad \forall i \in \{1, 2, \ldots, m\} \tag{2.3}$$

$$(a_1, a_2, \ldots, a_m) \notin S \tag{2.4}$$

2.1.2 Branching on a Set of Binary Columns

Vance et al. [1994] pointed out the following problem of conventional branching using a single variable. The branching rule does not divide the set of feasible solutions into two subsets of approximately equal size. The reason is as follows. The division of feasible solutions may be thought of equivalent to the division of the set of new columns that may get generated by the subproblem. On the right branch, the variable is lower bounded and the corresponding residual problem is generally smaller. Since the demand vector is reduced considerably, number of possible new columns that may be generated by the subproblem is small. However, on the left branch, since the variable is upper-bounded, we can not think of such a residual problem. Only one pattern is excluded, leading to a problem that is restricted not more than the one at the parent node.

Vance et al. [1994] introduced a new scheme of branching for the binary cutting stock problem (BCS). Here, the demand for each item is 1 i.e. $b_i = 1\,\forall i$. Also, they modified the Gilmore-Gomory formulation by making all inequality constraints as equality constraints. The formulation is the following:

$$\min \quad z = \sum_{j \in J} x_j \tag{2.5}$$

$$\text{s.t.} \quad \sum_{j \in J} a_{ij} x_j = b_i, \qquad i = 1, 2, \cdots m \tag{2.6}$$

$$x_j \text{ integer and } \geq 0, \qquad \forall j \in J \tag{2.7}$$

We claim that, this change in formulation does not alter the optimal integer solution. For that, we prove the following proposition.

Proposition 2.1.1 *Let x' be an optimal integer solution to the Gilmore-Gomory formulation and x'' be an optimal integer solution to the modified formulation. Then $\sum_{j\in J} x'_j = \sum_{j\in J} x''_j$.*

Proof Since every feasible solution to the modified formulation is also a solution to the Gilmore-Gomory formulation, $\sum_{j\in J} x'_j \leq \sum_{j\in J} x''_j$. Again, we can reduce items from columns selected by solution x' and get another set of columns x''' such that number of columns does not change though it becomes feasible for the modified formulation. Thus $\sum_{j\in J} x'_j = \sum_{j\in J} x'''_j \geq \sum_{j\in J} x''_j$. This completes the proof. ∎

Branching Scheme

The branching rule is based on the following proposition:

Proposition 2.1.2 (Ryan and Foster [1981]) *If any basic solution to the master LP relaxation for BCS is fractional, then there exists rows k and l of the master problem such that*

$$0 < \sum_{j:a_{kj}=1, a_{lj}=1} x_j < 1 \tag{2.8}$$

Proof Let $x_{j'}$ be a fractional variable and k be any row with $a_{kj'} = 1$. Let us consider the kth constraint of the form (2.6). We have $\sum_{j\in J} a_{kj}x_j = b_k = 1$. Now $a_{kj'}x_{j'}$ is one of the terms in the left hand side with $a_{kj'} = 1$ and x_j fractional. Since all the coefficients are either 0 or 1, there must be at least one more term $a_{kj''}x_{j''}$ such that $a_{kj''} = 1$ and $x_{j''}$ fractional. Now consider the two columns j' and j''. Since they are different and have 1 at row k, there must be one row l where they differ. Without loss of generality, assume that $a_{lj'} = 1$ and $a_{lj''} = 0$. Thus we have the following derivation.

$$\begin{aligned}
1 &= \sum_{j\in J} a_{kj}x_j \\
&= \sum_{j\in J|a_{kj}=1} x_j && \text{because coefficients are either 0 or 1} \\
&= \sum_{j\in J|a_{kj}=1,\, a_{lj}=1} x_j \;+ \sum_{j\in J|a_{kj}=1,\, a_{lj}=0} x_j \\
&> \sum_{j\in J|a_{kj}=1,\, a_{lj}=1} x_j
\end{aligned}$$

because $x_{j''}$ is among the omitted terms and we know it to be fractional and hence strictly greater than 0

$$> 0$$

because $x_{j'}$ is among the retained terms and we know it to be fractional and hence strictly greater than 0. ∎

Suppose, we find the pair k, l using proposition 2.1.2. Then, it is possible to branch on the set of variables $\{x_j \mid a_{kj} = 1, a_{lj} = 1\}$ by creating two new nodes

$$\sum_{j|a_{kj}=1,a_{lj}=1} x_j \leq 0 \text{ and } \sum_{j|a_{kj}=1,a_{lj}=1} x_j \geq 1 \tag{2.9}$$

The branching constraints have the following implication. On the left branch, all the columns having 1 at both the rows k and l must be set to zero. Further more, no column which satisfies this property should be generated by the subproblem. On the right branch the branching constraint along with the fact that only one column satisfies the demand of one item (because of the equality constraints in the formulation) implies that if any non-zero column has a 1 at one of the rows k and l must have 1 at the other row.

It can be seen that the branching scheme separates the solution space into subsets of approximately equal size. On the left branch only columns with 1 at only one of row k and row l or none, are valid. On the left branch only columns with 1 at both row k and row l or none, are valid. These two sets are approximately of equal size unlike in the case of branching on a single variable.

Modified Subproblem

We discuss the right branch first. There a, the new column generated, should satisfy $a_k = a_l$. This can be done by replacing the two items k and l by an item of width $w_k + w_l$ and profit $\pi_k + \pi_l$.

Solving the subproblem on the left branch, however, is trickier. There, the new column generated should not have 1 at both the rows k and l. This can be enforced by adding the constraint $a_k + a_l \leq 1$ to the subproblem.

However, at a node, deep in the branch and bound tree, there might be branching constraints of both types. So the subproblem formulation should have modified item sets (as done on the right branch) as well as several constraints of the form $a_k + a_l \leq 1$ (as done on the left branch). Let us assume that branching decision at each step is represented by the pair (k, l). Each node is associated with a set B of such pairs. Thus, in general, the subproblem looks like the following.

$$\max \quad \sum_{i=1}^{m'} \pi'_i a_i \tag{2.10}$$

$$\text{s.t.} \quad \sum_{i=1}^{m'} w'_i a_i \leq W \tag{2.11}$$

$$a_k + a_l \leq 1 \qquad \forall\,(k,l) \in B \tag{2.12}$$

$$a_i \in \{0,1\} \qquad \forall i = 1,\ldots,m' \tag{2.13}$$

where m' denotes the modified item count, π' and w' are the modified profit and weight vectors. We call this formulation Binary Knapsack Problem with Conflicting Item Pairs (BKPCIP).

2.1.3 Branching on a Set of Columns With Item Bounds

In the general cutting stock problem, the demand for items can possibly be more than 1. Hence, the column coefficients and right hand sides are general integers. The branching scheme used by Vance et al. [1994] based on the proposition 2.1.2 is not applicable when the master problem contains general integer values. Hence, the same branching rule can not be applied.

However, it seems that the basic spirit of the proposition 2.1.2 may be continued in the general CSP too. The key idea is to find a set of columns $\hat{J} \in J$ with a 'special property' such that the sum of the variables corresponding those columns is fractional, say equal to α. We can, then, have the following branching scheme.

$$\sum_{j\in\hat{J}} x_j \leq \lfloor\alpha\rfloor \qquad \text{and} \qquad \sum_{j\in\hat{J}} x_j \geq \lceil\alpha\rceil \tag{2.14}$$

Simultaneously, the 'special property' should also make sure that the modified structure of subproblem, as per this branching, still remains solvable. The modification is required because, the columns characterized by the 'special property' has fractional sum of values. However, in both the branches, we have added the constraints that the sum should not be fractional. We have to make sure that the subproblem does not generate the columns with this 'special property'. In this report, we describe two implementations of this idea. One of them, Vanderbeck [1999] is described in this subsection. The other, Vance [1998] is described in subsection 2.1.4.

Vanderbeck [1999] implemented branching based on a set of bounds on the number of instances of a particular item present in a column. We call such a bound as an *item bound constraint (IBC)* on a column. Let J be the set of columns present in the current master and $a \in J$ be such a column. An *item lower bound constraint* is of the form $a_i \geq v$.

The constraint is denoted by a triple $\beta \equiv (i, \geq, v)$, where $i \in 1, \ldots, m$ and $v \in \mathbb{N}$. Similarly an *item upper bound constraint* is of the form $a_i < v$ and denoted by $\beta \equiv (i, <, v)$. Let $J(\beta) = \{a \in J \mid a \text{ satisfies } \beta\}$. The complement of the IBC β, denoted by β^c, has the inequality $\geq$ replaced by $<$ and vice versa. It is easy to see that $J(\beta) \cap J(\beta^c) = \varnothing$ and $J(\beta) \cup J(\beta^c) = J$. Let B be a set of IBCs (possibly having either type). $J(B)$ is defined to be the set of columns that satisfy all IBCs in B, i.e., $J(B) = \cap_{\beta \in B} J(\beta)$.

Vanderbeck [1999] showed that given any master fractional solution, it is possible to find a set of IBCs B such that the columns which satisfy all the constraints in B, have fractional sum. Moreover, the cardinality of B is small. This fact is stated in proposition 2.1.3. We use the notation $f(B)$ to denote the sum of fractional parts of all columns satisfying all constraints of B, i.e., $f(B) = \sum_{j \in J(B)} x_j - \lfloor x_j \rfloor$. f is the sum of fractional parts of all columns in J. In an other way $f = f(\varnothing)$.

Proposition 2.1.3 *Given a fractional solution x to the master LP relaxation, there exists a set of IBCs B of size at most $(\lfloor \log f \rfloor + 1)$ such that $\sum_{j \in J(B)} x_j$ is fractional.*

Before going to the proof, let us think about the significance of the proposition. One could argue that without finding any IBCs, we get the set of columns (trivially all columns in the current master) whose sum is fractional (provided we have not got an solution such that the sum is integer). But in that case, if we use the set of columns for branching, then we get the worst branching. This is because all the integer feasible solutions are on one side of the branch. On the other hand, one could go to the other extreme of using a single fractional column and finding out set of IBCs, B, such that only that column satisfies B, and branch accordingly. However, in that case cardinality of B might be huge. Since the computation for branching, both in the master and the subproblem, depends on $|B|$, it is better to have a small B with fractional $\sum_{j \in J(B)} x_j$. Proposition 2.1.3 addresses that need.

Let us now have the proof. The proof is by construction. It is based on the following lemma which helps us find out the set of IBCs with smaller cardinality.

Lemma 2.1.4 *Suppose, we are given a set of IBCs B such that $f(B) \geq 1$. There exists a IBC $\beta \notin B$ such that $0 < f(B \cup \{\beta\}) \leq f(B)/2$.*

Proof of Lemma 2.1.4 Since $f(B) \geq 1$, there exist at least two columns $a_{*j_1}, a_{*j_2} \in J(B)$ such that $x_{j_1} - \lfloor x_{j_1} \rfloor > 0$ and $x_{j_2} - \lfloor x_{j_2} \rfloor > 0$. Since the two columns are different, there must be a row on which they differ. Let $r \in \{1, \ldots, m\}$ be that row. Without loss of generality let $a_{rj_1} < a_{rj_2}$. Let $v = \lceil (a_{rj_1} + a_{rj_2})/2 \rceil$. Consider the IBC $\beta \equiv (r, <, v)$ and its complement $\beta^c \equiv (r, \geq, v)$. Then $a_{*j_1} \in J(B \cup \{\beta\})$ and $a_{*j_2} \in J(B \cup \{\beta^c\})$. Thus $f(B \cup \{\beta\}) \geq x_{j_1} - \lfloor x_{j_1} \rfloor > 0$ and $f(B \cup \{\beta^c\}) \geq x_{j_2} - \lfloor x_{j_2} \rfloor > 0$. Again, since $f(B) = f(B \cup \{\beta\}) + f(B \cup \{\beta^c\})$, the smaller of $f(B \cup \{\beta\})$ and $f(B \cup \{\beta^c\})$ has value $\leq f(B)/2$. The corresponding IBC i.e. $B \cup \{\beta\}$ or $B \cup \{\beta^c\}$ is the required IBC. ∎

Proof of Proposition 2.1.3 Here we mention the idea without giving a formal proof. We start with empty B. As we keep on using Lemma 2.1.4 to add new IBCs, the f value reduces by half. Thus after at most $(\lfloor \log f \rfloor + 1)$ steps, f value reduces to less than 1 which is definitely fractional. Since, in each step one IBC is added, $|B|$ is upper bounded $(\lfloor \log f \rfloor + 1)$. In general, we may get B with fractional $f(B)$ early. However, we can not prove it. This is because, in the proof of Lemma 2.1.4, we show $f(B \cup \{\beta\})$ or $f(B \cup \{\beta^c\})$ to be greater than 0 but not necessarily fractional. ∎

Branching Scheme

The proposition 2.1.3 provides the foundation for branching. Suppose, we have got the set of IBCs B and the columns $J(B)$ that satisfy all constraints in B. Let $\alpha = \sum_{j \in J(B)} x_j$. We create two branches by adding branching constraint $\sum_{j \in J(B)} x_j \leq \lfloor \alpha \rfloor$ to the master LP on the left branch and $\sum_{j \in J(B)} x_j \geq \lfloor \alpha \rfloor + 1$ on the right. We call these branching constraints as of type-G and type-H respectively. It should be noted that these branching constraints are different from the IBCs. The branching constraints are added to master LP. However, IBCs are on the column elements and help find out the set of columns on which the branching happens. The link between them is that each of the branching constraints is associated with a set of IBCs.

Let us, now, think about the modified master LP relaxation at some node u in the branch-and-bound tree. Depending upon the path from root node to it, it will have some type-G and some type-H constraints. Let us partition the branching constraints into sets G^u and H^u according to their types. Thus the master LP relaxation at u takes the form

$$\min \quad \sum_{j \in J} x_j \tag{2.15}$$

$$\text{s.t.} \quad \sum_{j \in J} a_i x_j \geq b_i \qquad i = 1, \ldots, n \tag{2.16}$$

$$\sum_{j \in J(B_C)} x_j \leq K_C \qquad \forall C \in G^u \tag{2.17}$$

$$\sum_{j \in J(B_C)} x_j \geq L_C \qquad \forall C \in H^u \tag{2.18}$$

$$x_j \geq 0 \qquad j \in J \tag{2.19}$$

where C denotes a branching constraint. If it is of type-G, its right hand side is denoted by K_C; otherwise, by L_C. B_C denotes the set of IBCs associated with C.

Modified Subproblem

It can be noted that the master LP has more constraints. In fact, it has $m + |G^u| + |H^u|$ number of constraints. If a new column is generated, it should have $m + |G^u| + |H^u|$ entries. The first m, as in root node LP, are for the number of instances of the items included in the pattern corresponding to the column. However, the rest are 0/1 values such that they would have been the coefficients of the branching constraints in the column if the pattern were already present in the master LP. If we look for a column of negative reduced cost, the objective of the column generation subproblem takes the following form:

$$\max \quad \sum_{i=1}^{m} \pi_i a_i - \sum_{C \in G^u} \mu_C g_C + \sum_{C \in H^u} \nu_C h_C \tag{2.20}$$

where $(\pi, \mu, \nu) \in \mathbb{R}^{m+|G^u|+|H^u|}$ is an optimal dual solution to the master LP. Negative coefficients are due to the opposite direction of inequalities in G^u.

Let us now consider the constraints of the subproblem. The constraint on a_is is the usual knapsack capacity constraint. In addition, we have to make sure that if the new pattern satisfy all the IBCs associated with the branching constraint C, then the corresponding entry in the new column, (g_C or h_C which actually represents the coefficient of the branching constraint C in that column if it were already present in the master LP), must be 1; 0 otherwise.

Thus the constraints in subproblem look lie the following

$$\sum_{i=1}^{m} a_i w_i \leq W \tag{2.21}$$

$$g_C = \begin{cases} 1, & \text{if } a \in J(B_C) \\ 0, & \text{otherwise} \end{cases} \qquad \forall\, C \in G^u \tag{2.22}$$

$$h_C = \begin{cases} 1, & \text{if } a \in J(B_C) \\ 0, & \text{otherwise} \end{cases} \qquad \forall\, C \in H^u \tag{2.23}$$

However, we need to express the logical constraint $a \in J(B_C)$ using one or more linear constraints. For that, we express the logical constraint as logical AND of smaller constraints as follows

$$a \in J(B_C) \Longleftrightarrow \wedge_{\beta \in B_C}(a \in J(\beta)) \Longleftrightarrow \wedge_{(i,\geq,v) \in B_C}(a[i] \geq v) \bigwedge \wedge_{(i,<,v) \in B_C}(a[i] < v)$$

For each of the smaller logical constraint $\beta \in B_C$ we need to use a variable $\eta_\beta \in \{0, 1\}$. Then we need to find the linear relationship among g_C (h_C) and the η_βs. However, since the signs of objective of g_C and h_C are different, we derive the corresponding linear constraints differently.

For g_C the objective is negative. So if we can provide a lower bound on g_C it will be enough, because the solver will anyway try to make it as low as possible so that the objective is maximized. We add the lower bound as follows

$$g_C \geq 1 - \sum_{\beta \in B_C} (1 - \eta_\beta)$$

The lower bounds on the variables for smaller constraints are added as follows

$$\begin{aligned} (a_i^{\max} - v + 1)\eta_\beta &\geq (a_i - v + 1) && \forall \beta \equiv (i, \geq, v) \in B_C \\ v\,\eta_\beta &\geq v - a_i && \forall \beta \equiv (i, <, v) \in B_C \end{aligned}$$

where $a_i^{\max} = \min\{b_i, \lfloor \frac{W}{w_i} \rfloor\}$ denotes the maximum possible value of a_i.

On the other hand, it is enough to provide an upper bound on h_C. We add the upper bound as follows

$$h_C \leq \eta_\beta \qquad \forall\, \beta \in B_C$$

The upper bounds on the variables for smaller constraints are added as follows

$$\begin{aligned} v\,\eta_\beta &\leq a_i && \forall \beta \equiv (i, \geq, v) \in B_C \\ (a_i^{\max} - v + 1)\eta_\beta &\leq a_i^{\max} - a_i && \forall \beta \equiv (i, <, v) \in B_C \end{aligned}$$

Thus the consolidated form of the subproblem is the following

$$\max \quad \sum_{i=1}^{m} \pi_i a_i - \sum_{C \in G^u} \mu_C g_C + \sum_{C \in H^u} \nu_C h_C \tag{2.24}$$

$$\text{s.t.} \quad \sum_{i=1}^{m} a_i w_i \leq W \tag{2.25}$$

$$g_C \geq 1 - \sum_{\beta \in B_C} (1 - \eta_\beta) \qquad \forall B_C \in G^u \tag{2.26}$$

$$(a_i^{\max} - v + 1)\eta_\beta \geq (a_i - v + 1) \qquad \forall \beta \equiv (i, \geq, v) \in B_C,\ \forall B_C \in G^u \tag{2.27}$$

$$v\eta_\beta \geq v - a_i \qquad \forall \beta \equiv (i, <, v) \in B_C,\ \forall B_C \in G^u \tag{2.28}$$

$$h_C \leq \eta_\beta \qquad \forall \beta \in B_C,\ \forall B_C \in H^u \tag{2.29}$$

$$v\eta_\beta \leq a_i \qquad \forall \beta \equiv (i, \geq, v) \in B_C,\ \forall B_C \in H^u \tag{2.30}$$

$$(a_i^{\max} - v + 1)\eta_\beta \leq a_i^{\max} - a_i \qquad \forall \beta \equiv (i, <, v) \in B_C,\ \forall B_C \in H^u \tag{2.31}$$

$$a_i \in \{0, 1, \ldots, a_i^{\max}\} \qquad i = 1, \ldots, m \tag{2.32}$$

$$\eta_\beta \in \{0, 1\} \qquad \forall \beta \in B_C,\ \forall B_C \in G^u \cup H^u \tag{2.33}$$

Simpler Branching Scheme

A simpler branching scheme can be designed if the columns of the master LP are stored as binary columns. This is because, if the column elements are binary, the IBCs take the form $\beta \equiv (i, \geq, 1)$ or $\beta \equiv (i, <, 1)$. This is equivalent to saying that $\beta \equiv (a_i = 1)$ or $\beta \equiv (a_i = 0)$. This can be denoted in short as $\beta \equiv (i, 1)$ or $\beta \equiv (i, 0)$. This also simplifies the way the subproblem is formulated.

Before going to further details, let us see how binary columns can be obtained. The binary equivalent $a' \in \{0,1\}^{m'}$ of a general column $a \in \mathbb{N}^m$ can be obtained by using the conversion as follows. Let, $m' = \sum_{i=1}^{m} m_i$, where $m_i = \lceil \log(a_i^{\max} + 1) \rceil$ and $a_i^{\max}$ denotes the maximum possible value of a_i. The elements of a and a' have the relation $a_i = \sum_{l=0}^{m_i - 1} 2^l \, a'_{p_i + 1}, \ \forall i = 1, \ldots, m$, where $p_i = 1 + \sum_{l=1}^{i-1} m_l$. The equivalence between a and a' is represented by $a \leftrightarrow a'$. To avoid confusion, from now on, we will use i to index a and l to index a'.

We need to extend some of the symbols used earlier to be applied on the converted binary columns. Let B be a set of IBCs on the binary columns. $J(B)$ is defined to be the set of columns in the master LP, that satisfy all IBCs in B, i.e., $J(B) = \{a \in J \mid a \leftrightarrow a', \ a'_l = 0 \, \forall (l, 0) \in B, \text{ and } a'_l = 1 \, \forall (l, 1) \in B\}$.

With this new definition of IBCs, Proposition 2.1.3 still holds. However, the proof in Lemma 2.1.4 need to be slightly modified for this. Let the binary representation of the two columns used there be the following. $a_{*j_1} \leftrightarrow a'_{*j_1}, a_{*j_2} \leftrightarrow a'_{*j_2}$. Since the two columns are different, there must be a row in the binary representation, on which they differ. Let $l \in \{1, \ldots, m'\}$ be that row. Without loss of generality let $a'_{lj_1} = 0$ and $a'_{lj_2} = 1$. Then $\beta \equiv (l, 0)$ and $\beta^c \equiv (l, 1)$. The rest of the proof remains same.

Let us now think about the branching. The scheme and master LP formulation at a node u after branching remain same because the change is encapsulated in the way the IBCs $C \in B_C, \ \forall \, B_C \in G^u \cup H^u$ are stored.

Simpler Modified Subproblem

As before, the objective of the column generation subproblem have the form:

$$\max \quad \sum_{i=1}^{m} \pi_i a_i - \sum_{C \in G^u} \mu_C g_C + \sum_{C \in H^u} \nu_C h_C \tag{2.34}$$

However, since all the IBCs are on the binary representation of the column, we change the pattern a to its equivalent binary form a'. The objective and item weights are changed as follows. $\pi'_l = 2^k \pi_i, \ w'_l = 2^k w_i, \ \forall l = p_i + k, \ k = 0, \ldots, m_i - 1$, and $i = 1, \ldots, m$. Thus

the modified objective is

$$\max \quad \sum_{l=1}^{m'} \pi'_l a'_l - \sum_{C \in G^u} \mu_C g_C + \sum_{C \in H^u} \nu_C h_C \tag{2.35}$$

Now we add constraints for the IBCs. For each of the smaller logical constraint $\beta \in B_C$ we do not need extra variables η_β. For $\beta \equiv (l, b), b \in \{0, 1\}$, we can directly use a'_l. As done previously, we derive the corresponding linear constraints for g_C and h_C differently. We provide a lower bound on g_C as follows

$$g_C \geq 1 - \sum_{(l,0) \in B_C} a'_l - \sum_{(l,1) \in B_C} (1 - a'_l)$$

On the other hand, we add an upper bound on h_C as follows

$$\begin{aligned} h_C &\leq (1 - a'_l) && \forall\, (l, 0) \in B_C \\ h_C &\leq a'_l && \forall\, (l, 1) \in B_C \end{aligned}$$

Thus, the subproblem takes the following final form

$$\max \quad \sum_{l=1}^{m'} \pi'_l a'_l - \sum_{C \in G^u} \mu_C g_C + \sum_{C \in H^u} \nu_C h_C \tag{2.36}$$

$$\text{s.t.} \quad \sum_{l=1}^{m'} w'_l a'_l \leq W \tag{2.37}$$

$$\sum_{l=p_i}^{p_i+m_i-1} 2^{l-p_i} a'_l \leq a_i^{\max} \qquad \forall i = \{1, \ldots, m\} \tag{2.38}$$

$$g_C \geq 1 - \sum_{(l,0) \in B_C} a'_l - \sum_{(l,1) \in B_C} (1 - a'_l) \qquad \forall C \in G^u \tag{2.39}$$

$$h_C \leq (1 - a'_l) \qquad \forall\, (l, 0) \in B_C,\ \forall C \in H^u \tag{2.40}$$

$$h_C \leq a'_l \qquad \forall\, (l, 1) \in B_C,\ \forall C \in H^u \tag{2.41}$$

$$a'_l \in \{0, 1\} \qquad \forall l = \{1, \ldots, m'\} \tag{2.42}$$

2.1.4 Branching on a Set of Columns with Item Lower Bounds

In subsection 2.1.3, we saw a branching scheme based on a set of item bound constraints (IBCs). There, the IBCs were of both lower and upper bounds. However, it seems that using lower bounds only, it is possible to develop branching. The branching scheme proposed in Vanderbeck and Wolsey [1996] is such a special case of the scheme described in 2.1.3. In this scheme the set of IBCs B consists of IBCs of the form $\beta \equiv (i, \geq v)$.

However, the size of B may be larger than $\lfloor \log f \rfloor + 1$.

Vance [1998] implemented a branching scheme which is a special case of using item lower bound constraints (ILBCs). Before going further, let us restate the scheme using ILBCs. We find a set of ILBCs B such that

$$\sum_{j \mid \forall (i,\geq,v) \in B,\, a_{ij} \geq v} x_j = \alpha \tag{2.43}$$

is fractional and branch accordingly. Now we discuss the implementation in Vance [1998].

Branching scheme

In the implementation, Vance [1998], assumed that only maximal cutting patterns are allowed in the master problem. By maximal, we mean that the waste left after cutting this pattern is shorter than the width of the smallest item. In other words, no more item can be added in the pattern. We now prove that this assumption does not alter optimal integer solution to the problem.

Proposition 2.1.5 *Let x' be an optimal integer solution to the Gilmore-Gomory formulation and x'' be an optimal integer solution to the formulation restricted with maximal columns. Let J' and J'' be the set of columns respectively. $J'' \subseteq J'$. Then $\sum_{j \in J'} x'_j = \sum_{j \in J''} x''_j$.*

Proof Since every feasible solution to the restricted formulation is also a solution to the Gilmore-Gomory formulation, $\sum_{j \in J'} x'_j \leq \sum_{j \in J''} x''_j$. Now suppose, there is a column in x'' which is not maximal. Then, we can go on adding instances of the smallest item till we get an maximal column. Thus get another set of columns x''' such that number of columns does not change though it becomes feasible for the modified formulation. Thus $\sum_{j \in J'} x'_j = \sum_{j \in J''} x'''_j \geq \sum_{j \in J''} x''_j$. This completes the proof. ∎

Vance [1998] utilized another fact from Vanderbeck and Wolsey [1996] that if only maximal columns are used in the master problem, any fractional column can define the set of ILBCs B. It can be explained as follows. Suppose, x_k is a fractional column. The B consists of all ILBCs of the form $(i, \geq, a_{ik})$ such that $a_{ik} > 0$. Since x_k is maximal, no other column satisfies all ILBCs in B. Thus $\sum_{j \mid \forall (i,\geq,v) \in B,\, a_{ij} \geq v} x_j = x_k$ is fractional.

Modified Subproblem

It can be noted that the branching scheme is the same as the conventional branching where only a single variable is used for branching. Only difference is that the columns are maximal. Thus, the subproblem takes the form of a BKPFS (see 2.1.1) with extra

condition that only maximal configurations are valid. We call this form of the subproblem Bounded Maximal Knapsack Problem with Forbidden Solutions (BMKPFS).

The fact that the columns are maximal makes solving the subproblem easier. We discuss the advantage of using maximal columns in solving the subproblem in subsection 2.2.

This brings us to the end of the discussion on the different branching schemes we studied. We now describe the different approaches to solve the subproblem.

2.2 Solution to Modified Subproblem

In section 2.1, we saw that depending upon the branching scheme, we need to solve the subproblem in one the following forms

- bounded knapsack problem with forbidden solutions (BKPFS), as discussed in subsection 2.1.1
- binary knapsack problem with conflicting item pairs (BKPCIP), i.e., with constraints of the form $a_k + a_l \leq 1$, discussed in subsection 2.1.2
- general integer programming problem with sets of item bound constraints that does not have any special structure (so far unknown), discussed in subsection 2.1.3
- BKPFS with the extra constraint of maximal configurations (BMKPFS), discussed in 2.1.4

One of them can not be solved with out general integer programming problem solver. The others have special structure and can be solved using specialized algorithms. We discuss the solvers that we studied.

Solution to BKPFS

In general, the problem is solved by modifying the branch-and-bound based algorithm implemented by Horowitz and Sahni [1974] that solves a bounded knapsack algorithm without any forbidden set. We call the implementation as HS1 algorithm. We, first, state the algorithm and then modify it to work with forbidden set too.

The HS1 algorithm enumerates the solution space by considering items in the order of non-increasing profit density (π_i/w_i). This order is well known as 'greedy order'. However, before exploring a branch, it calculates a upper bound on the profit on that branch. If the bound is not greater than the best solution known so far, it does not explore that branch. The bound is calculated as follows. Let at the current node in the enumeration tree, the algorithm has already fixed items $1, \ldots, s$ where $s < m$ with the values $\hat{a}_1, \ldots, \hat{a}_s$. An

upper bound, based on the LP bound, corresponding to the current partial solution $\hat{a}$ is given by

$$U = \sum_{i=1}^{s} \pi_i \hat{a}_i + \left(W - \sum_{i=1}^{s} w_i \hat{a}_i \right) \pi_{s+1}/w_{s+1} \tag{2.44}$$

A pseudocode for the algorithm is given in Algorithm 1.

Algorithm 1: HS Algorithm

Input : An instance $(m; W; w; \pi; b)$ of the bounded knapsack problem
Output: An optimal solution vector a

```
1  Initialize: s = 1, M = 0, â_i = 0, for all i = 1, ..., m
2  repeat
3      if (M < U(â)) then
4          s = s + 1
           /* Update partial solution                                  */
5          â_s = min{b_s, (W − Σ_{i=1}^{s−1} w_i â_i)/w_s}
6          If â is an improvement, save it in a
           /* Backtrack to next unfathomed soln in the enumeration
              order                                                    */
7          Find maximum k ≤ s such that â_k > 0,
8          s = k
9          â_s = â_s − 1
10 until (s == 1) ;
```

To work with forbidden set, the algorithm is modified as follows. Just before updating a by a better solution $\hat{a}$, it is searched in the forbidden set. a is updated only when the search fails, i.e., $\hat{a}$ is not in the forbidden set. The modified algorithm is called HS2.

Solution to BKPCIP

Vance et al. [1994] replaces the constraints due to the conflicting item pairs by a graph. The graph has a node for each of the items and an edge between two the nodes which are conflicting. We call this graph the constraint graph. Without the constraints, the subproblem takes the form of the standard 0/1 knapsack problem. However, we have to modify the knapsack solver such that the solution does not contain conflicting items. Vance et al. [1994] shows that an efficient implementation for such a modification is possible in the special case where the edges in the constraint graph are non-overlapping (without common node). However, if the special property is not met, Vance et al. [1994] solves the problem using a general integer programming solver. We now describe the implementation in the special case.

When there is no overlapping edge, the problem can be formulated as a Binary Knapsack

Problem with Special Ordered Sets (BKPSOS). A special ordered set (SOS) is a set of variables among which at most one can be non zero. The formulation is as follows:

$$\max \quad \sum_{j\in L}\sum_{i\in S_j} \pi_{ij}\, x_{ij} \tag{2.45}$$

$$\text{s.t.} \quad \sum_{j\in L}\sum_{i\in S_j} w_{ij}\, x_{ij} \le W \tag{2.46}$$

$$\sum_{i\in S_j} x_{ij} \le 1 \qquad \forall j \in L \tag{2.47}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in S_j,\ j \in L \tag{2.48}$$

where S_js, $j \in L$ are the SOSs. Each edge constraint is represented by a SOS of size 2 and each disconnected nodes is represented by SOSs of size 1.

Vance et al. [1994] used a modified version of the HS1 algorithm (see Algorithm 1) to solve the BKPSOS. We call this modified algorithm HS3. The modification is as follows:

- The greedy ordering of the variables is replaced by an ordering of the SOSs as defined below:

$$\frac{\pi_{i_1}}{w_{i_1}} \ge \frac{\pi_{i_2}}{w_{i_2}} \ge \cdots \ge \frac{\pi_{i_L}}{w_{i_L}} \qquad \text{where,} \qquad \frac{\pi_{i_j}}{w_{i_j}} = \max\left\{\frac{\pi_{ij}}{w_{ij}} \mid i \in S_j\right\} \forall\, j \in L \tag{2.49}$$

- Enumeration is done by adding at most one item from each SOS to the current solution.

- The upper bound is calculated using the LP relaxation [1] of the residual BKPSOS as follows

$$U = \sum_{j=1}^{s}\sum_{i\in S_j} \pi_{ij}\hat{a}_{ij} + JP(\{(s+1),\ldots,h\}, W - \sum_{j=1}^{s}\sum_{i\in S_j} w_{ij}\hat{a}_{ij}) \tag{2.50}$$

  $JP(S,W)$ denotes the LP relaxation of the BKPSOS containing the set of SOSs S and knapsack size W.

- While backtracking, before going back to the previous SOS set, the item inserted last is removed from the current solution and another item (if any) in the current SOS is considered (if it was not done already).

Solution to BMKPFS

The solution to the subproblem should address the following

[1] Vance et al. [1994] used an efficient algorithm, as given in Johnson and Padberg [1981], to get this value

1. the new patterns generated should be maximal
2. no forbidden configuration should be generated

To address these issues, Vance [1998] modified algorithm HS1 (see subsection 2.2). We call this modified algorithm HS4. The modification is as follows.

- The issue of generating maximal pattern is achieved by including items with zero profit too, provided it is not full enough. Since the items with zero profit are at the last in the greedy ordering, optimality is maintained.
- The same modification as in HS2 (see subsection 2.2) would suffice to make sure that no forbidden configuration is generated. However, it seems that we can do better in terms of efficiency. This is done as follows. For each forbidden pattern a, the item i with $a_i > 0$ which appears last in the greedy order is marked. It is done before the actual enumeration begins. During enumeration, if a marked item is set some value, it is immediately checked in the forbidden set. If present we can backtrack immediately, there by, avoiding some of the unnecessary enumerations. This scheme is applicable only when the patterns are maximal.

In summary, we described the algorithms for the subproblems formulated in section 2.1. Next we describe another subtask of a branch-and-price algorithm which is related to branching.

2.3 Node Selection

If the conventional branching on a single variable is used (section 2.1.1), the subproblem on the right branch is easier to solve. Also, there is a possibility of reaching to a solution quickly because the residual problem on the right branch is considerably smaller than the original problem. Also, because of the fact that most of the 1D-CSP satisfy IRUP property, the LP bound found on a node is strong enough to go to the depth of a branch. For these reasons, almost all the branching schemes based on a single variable, e.g. Degraeve and Schrage [1999], Degraeve and Peeters [2003], Vance [1998], used dfs with higher priority to the right branch. Belov and Scheithauer [2006] uses branching on a single variable. However, it implements the hybrid approach of diving from a bfs node. At the beginning some number of nodes are chosen according bfs and after that their children are processed in the dfs fashion. This approach helps in solving Non-IRUP instances.

Though, Vance et al. [1994] implements branching on a set of columns, the observations on node selection made in the context of single variable branching (i.e. easier subproblem, quick solution to the constrained problem on the right branch), are still applicable. For this reason, it implemented dfs with higher priority to right branch.

The branching scheme of Vanderbeck [1999], by design, does not have imbalance between the two branches. However, because of the IRUP property, it is better to go in depth of a branch. Thus, dfs with equal priority to both the children was implemented.

However, it can be noted that the selection of nodes are not the only choices that are made in the algorithms. There may be different parameters depending upon the branching scheme. For example, in case of branching on single variables, if there are multiple fractional variables which one should be picked? It is the general practice to choose the variable having largest distance from nearest integer values on both ends. That is, $x_1 = 1.5$ is given higher priority than $x_2 = 3.05$. This helps in better separation. However, for other parameters, the decision may not be easy. For example, if there are different sets of IBCs B in case of branching on multiple columns, which one should be used? This needs deeper study of the implementation which we have not undertaken.

2.4 Heuristic Solutions

The performance of a branch-and-price based algorithm depends on the quality of the initial heuristic solution used at the root node as well as the heuristic solution obtained by rounding the LP solution at any node in the branch and bound tree. If this heuristic solution is good, we can quickly fathom the nodes whose LP relaxations are higher than this heuristic solution. In addition, the heuristic solution at the root node helps find initial set of feasible columns.

2.4.1 Common Rounding Heuristics

The most commonly used heuristic solution is the First Fit Decreasing (FFD) heuristic. In this scheme the items are filled in a non-increasing order of their widths. The item under consideration is tried to fit in a stock used already. If it can not be done so, a new stock is used. While searching for space in the stocks already used, the stocks are searched in the order they were used first. Another heuristic procedure, Best Fit Decreasing (BFD), is similar to the FFD. The only difference is that while searching for space in the already used stocks, the one with smallest unused part is picked.

However, the performance of these procedures are not good in practice. There are different enhanced heuristics. We studied one such procedure, Sequential Value Correction (SVC) method, which is described in section 2.4.2. The method constructs an heuristic solution from the LP solution. However, the two heuristics described here are still useful in finding the initial solution when no other information such as LP solution etc. are known.

2.4.2 A Sequential Heuristic Approach

The branch-and-price based algorithm described in Belov and Scheithauer [2006] uses a sequential value correction (SVC) heuristic. Like the other algorithms, it first obtains rounded down LP solution and then solves the residual problem. However, unlike others, it constructs a solution for the residual problem by generating pattern after pattern. While solving iteratively, SVC constructs a new pattern using some information from previous patterns. This information, *pseudo-prices* of products, is, in concept, similar to the dual multiplier of the master LP relaxation. It is calculated on the basis of the material-per-item-type consumption. Let $l = W - \sum_{i=1}^{m} w_i a_i$ be the pattern waste. $W - l$ represents the utilization of the pattern. The expression $\frac{W}{W-l}$ reflects how 'bad' the current pattern is. For a pattern with 0 waste, it is the minimum 1. The more the waste, the bigger the expression is. Now the expression $\frac{W}{W-l} a_i w_i$ is a measure of the material-per-item-type consumption by item i or, rather, 'badness' contributed by it. Suppose, we have already generated n patterns. The expression $\sum_{j=1}^{n} \frac{W}{W-l_j} a_{ij} w_i$ gives a measure of the 'badness' of item i with respect to the current set of generated patterns. However, if the demand of an item is more, and the corresponding expression is more, we should not penalize that item much. So the expression $\frac{1}{b_i} \sum_{j=1}^{n} \frac{W}{W-l_j} a_{ij} w_i$ gives a better measure of the 'badness' of item i. Now, it makes sense to satisfy the demands of the 'bad' items as soon as possible in the sequential heuristic process so that we can try with the other 'good' items which mixes easily with each other to generate good patterns. So, this expression is used as the cost function for generating a new pattern.

The implementation of Belov and Scheithauer [2006] maintains the pseudo-prices $y \in R_+^m$ as follows. They are initialized with scaled simplex multipliers. After generating pattern a maximizing $y^T a$, they are 'corrected'. In this implementation, $\frac{W}{W-l} a_i w_i^p$ is taken as a measure of the 'badness' contributed by item i, where p is a parameter slightly greater than 1. Experimentally, $p = 1.03$ was found to be a 'good' value. The new weight of piece i is the following weighted average:

$$y_i \leftarrow g_1 y_i + g_2 \frac{W}{W-l} w_i^p, \qquad \forall i : a_i > 0 \tag{2.51}$$

where g_1, g_2 are update weights with

$$g_1/g_2 = \Omega(b_i' + b_i'')/a_i, \tag{2.52}$$

$g_1 + g_2 = 1$, b_i' is the residual demand of item i after the rounding down procedure, b_i'' is the so far unmet demand of the item i. The old value is weighted more if the total demand of the item is more. Otherwise the new value is given more weight. The value of the randomization factor Ω is uniformly chosen from $[1/\bar{\Omega}, \bar{\Omega}]$ before generating each pattern. $\bar{\Omega}$ is uniformly chosen from $[1, \bar{\bar{\Omega}}]$ for each new solution. $\bar{\bar{\Omega}} = 1.5$ was found to be a 'good' value.

The heuristic solution is given in Algorithm 2. At the initialization step, the dual multipliers are modified to some small value if they are zero. This is because, the oversupplied items in the LP solution have dual value zero. However they may become undersupplied in the rounding-down step. Without this modification, those items may never be picked by the pattern generator.

This heuristic is repeated multiple times and the best solution is picked. This is done in the outer loop. The optimality can be proved by using the rounded up value of residual LP solution as a lower bound.

Algorithm 2: Sequential Value Correction

Input : An instance $(m; W; w; b')$ of 1D-CSP; The simplex multipliers $\pi_1, \ldots, \pi_m$

Output: A feasible solution vector $(x_{a_{*j}}), j \in \{1, \ldots, n\}$

Initialize: $y_i = \max\{1, W\pi_i\}$ for all $i = 1, \ldots, m,\ k = 0$
repeat
  $b'' = b'; \tilde{x} = 0; k = k + 1$; `/* start new solution */`
  repeat
    $a = \arg\max\{y^T a : w^T a \leq W; a \leq b''; a \in Z^m\}$
    $f = \min_{i:a_i>0}\{\lfloor b''_i/a_i \rfloor\}$ `/* choose pattern frequency */`
    $\hat{x}_a = \hat{x}_a + f; b'' = b'' - fa$; `/* reduce the right-hand side */`
    Update Weights $y_i, 1 \leq i \leq m$; `/* value correction using 2.51 */`
  until $b''_i = 0$ *for all* $1 \leq i \leq m$;
  If $\hat{x}$ is an improvement, save it in x
until *(Optimality is Proved) or (Iteration Limit is Exceeded)* ;

2.5 Lower Bounds for Early Termination of Column Generation

In section 1.5.2 we mentioned about the problem of slow convergence of the column generation process and showed that there is a way to detect early in the column generation process if the LP solution for the current node will be useful at all. Let us restate the symbols used there. We used z^u_{IP}, z^u_{LP} and $\bar{z}^u_{LP}$ to denote the integer solution to the problem on the space corresponding to node u, the corresponding LP solution and the LP solution for the restricted master problem respectively. We also showed that if $\bar{c}$ is the objective of the subproblem solved last, then the quantity $\bar{z}^u_{LP}/\bar{c}$ can be used as a lower bound for z^u_{IP}. If the value is greater than the current incumbent the column generation is terminated.

Since, at the root node, the incumbent is not generally good, this test most often fails. Vance et al. [1994] and Vance [1998] used an additional trick to terminate the root node

column generation early. They used the following proposition.

Proposition 2.5.1 *If* $\lceil \bar{z}^r_{LP} \rceil = \lceil \bar{z}^r_{LP}/\bar{c} \rceil$, *then* $\lceil \bar{z}^r_{LP} \rceil \leq z^r_{IP}$ *where* r *denotes root node.*

Proof We already showed that $\bar{z}^r_{LP}/\bar{c} \leq z^r_{LP}$. Taking celling of both sides we get, $\lceil \bar{z}^r_{LP}/\bar{c} \rceil \leq \lceil z^r_{LP} \rceil$. Using the given assumption, we have $\lceil \bar{z}^r_{LP} \rceil \leq \lceil z^r_{LP} \rceil$. But we know that $\bar{z}^r_{LP}$ is an upper bound for z^r_{LP}. Hence, $\lceil \bar{z}^r_{LP} \rceil \geq z^r_{LP}$. Combining inequalities in both directions, we have $\lceil \bar{z}^r_{LP} \rceil = \lceil z^r_{LP} \rceil$. However, $\lceil z^r_{LP} \rceil \leq z^r_{IP}$. Thus, $\lceil \bar{z}^r_{LP} \rceil \leq z^r_{IP}$. ∎

The importance of the proposition is that if, during column generation, we have already reached a state where $\lceil \bar{z}^r_{LP} \rceil = \lceil \bar{z}^r_{LP}/\bar{c} \rceil$, then we can no longer improve the lower bound and hence the process can be terminated. This is because, from this onward the lower bound $\lceil z^r_{LP} \rceil$ can decrease only.

So far we have discussed some of the algorithms based on pure branch-and-price. Now we describe a branch-cut-price algorithm.

2.6 Cutting Plane Algorithm

Apart from the standard branch-and-price algorithms, there have been several efforts on solving the problem by using cutting plane algorithms. Here, we discuss an algorithm (Belov and Scheithauer [2006]) which uses a combination of both the approaches. In this method, the LP relaxation at each branch-and-price node is strengthened by applying cuts. The rest is similar to the standard branch-and-price algorithms. Branching is based on a single variable as described in subsection 2.1.1. The heuristic solutions are obtained using SVC method as described in subsection 2.4.2. Here we provide the basic information of the cuts that are used and then the aspects that different from the algorithms described so far.

In subsection 2.6.1 we describe the principle behind using super-additive cuts which are based on linear combinations of current constraints. Chvatal-Gomory cuts, Gomory fractional and Gomory Mixed Integer cuts are special super-additive cuts. We show in subsection 2.6.2 how to construct these special cuts for the Gilmore-Gomory formulation at the root node. At the internal nodes, the branching variables are upper/lower bounded. The modification necessary for this is discussed in subsection 2.6.3. The addition of the cuts makes the subproblem complex. We explain the modifications in the subproblem formulation in subsection 2.6.4 and how it is solved in subsection 2.6.5. We conclude the section with some remarks on the overall algorithm in the subsection 2.6.6

2.6.1 Gomory Fractional and Mixed-Integer Cuts

With respect to the current solution x of the LP relaxation $Ax = b, x \geq 0$, a cut is defined as an inequality

$$\sum_{j=1}^{n} F(u^T a_{*j}) x_j \leq F(u^T b), \tag{2.53}$$

where u is some vector producing a linear combination of the existing constraints and $F : \mathcal{R}^1 \rightarrow \mathcal{R}^1$ is some super-additive non-decreasing function such that the cut is valid for all feasible integer solutions but invalid for the current fractional solution. An example of F is the simple round down function $\lfloor . \rfloor$. There can be several ways of getting the multiplies u. One of them is shown in section 2.6.2. If the LP solution of the problem including this cut is not still integer, further inequalities are constructed recursively, based on linear combinations of original constraints and the cuts added already.

Suppose initially there were m constraints. Assume that we have generated $r-1$ cuts already. The coefficient of the cut r in column j is defined recursively as follows:

$$\psi_j^r = F(\sum_{i=1}^{m} u_i^r a_{ij} + \sum_{t=1}^{r-1} u_{m+t}^r \psi_j^t), \quad r = 1, \ldots, \mu,$$

where $u_i^r, i = 1, \ldots, m+r-1$, are the coefficients of the linear combination of all previous constraints. It can be noted that ψ_j^r can be thought of as a function of the column a_{*j}. This is because, if we unfold the last part of the right hand side by applying the definition recursively, all the ψ terms will vanish and only an expression with a_{ij}s and u_i^rs will remain. Thus we also write $\psi_j^r = \psi^r(a_{*j})$.

The rth cut takes the form

$$\sum_{j=1}^{n} \psi_j^r x_j \leq \psi_0^r$$

where $\psi_0^r = \psi^r(b)$.

Since each added cut is an inequality constraint, we need to add a slack variable to generate further cuts. $\sum_{j=1}^{n} \psi_j^r x_j + s = \psi_0^r,\ s \geq 0$. The slack variable may or may not be integer depending on the function F is used. Further cuts which depend on this cut have their own coefficients in this column. Thus, cut r in general can be represented as

$$\sum_{j=1}^{n} \psi_j^r x_j + \sum_{t=1}^{r-1} \psi^r(s_v) s_v + s_r = \psi_0^r. \tag{2.54}$$

The coefficient of slack variables are set to 1, i.e., $\psi^v(s_v) = 1, v = 1, \ldots, \mu$. If the

slack variables are integers, they can be treated similar to the original variables. The corresponding columns are in the form $(0, 0, \ldots, 1)$. Thus, the coefficient $\psi^r(s_v)$ of cut r for the slack variable of cut v that is generated already is given by $\psi^r(s_v) = F(\sum_{t=1}^{r-1} \psi^t(s_v))$, similar to (2.54).

Chvatal-Gomory valid inequalities are constructed using the rounding down function, $F(u^T a) = \lfloor u^T a \rfloor$. Because of this, the slack variables are integer and hence can be treated similar to the original variables.

Gomory fractional cuts are constructed by subtracting inequality $\lfloor u^T A \rfloor x \leq \lfloor u^T b \rfloor$ from $u^T Ax = u^T b$. This results in $\sum_{j=1}^{n} fr(u^T a_{*j}) x_j \geq fr(u^T b)$, where $fr(d) = d - \lfloor d \rfloor$. Here the slack variables are fractional and hence needs to be taken care of when subsequent cuts are added. This means, once a Gomory fractional cut is added, subsequently only mixed integer cuts can be added.

Gomory mixed-integer cuts are constructed using $F_\alpha(d) = \lfloor d \rfloor + \max\{0, (fr(d) - \alpha)/(1 - \alpha)\}$ which is a generalization of the rounding down function $\lfloor . \rfloor$.

With this basic introduction to the cuts, we now obtain the cuts for the problem we are concerned about.

2.6.2 Cutting Planes without Branching

Belov and Scheithauer [2006] used Chvatal-Gomory valid inequalities on the master LP relaxation of Gilmore-Gomory formulation after adding the slacks in the demand constraints for each of m items.

The only remaining part to be explained for the implementation of the cuts is how the multipliers us are selected. Belov and Scheithauer [2006] constructed them using the optimum simplex tableau. Suppose, the columns of the final simplex tableau is rearranged so that basis part is separated from the others, i.e., $A = (A_B, A_N)$. Let x be also similarly rearranged, i.e., $x = (x_B, x_N)$. Then, we have $A_B x_B + A_N x_N = b$, or $x_B + A_B^{-1} A_N x_N = A_B^{-1} b$. Suppose, the ith basic variable is fractional. Then the cut can be generated by rounding down both sides of row i. Suppose v and w denote the ith row of $A_B^{-1} A$ and $A_B^{-1} b$ respectively. That results in the cut $\lfloor v \rfloor x \geq \lfloor wb \rfloor$ or equivalently $-\lfloor v \rfloor x \leq \lfloor -wb \rfloor$. It can be easily shown that this cut removes the current LP optimum. Thus the required $u = -w$, i.e., the ith row of $-A_B^{-1}$.

2.6.3 Cutting Planes with Branching

There is a complication in applying the cutting planes at an internal node of the branch-and-bound tree. This is because, at an internal node some variables (which are selected in branching on a path from root to this node) are already set to lower and upper bounds as branching constraints. This is taken care of as follows.

Let the LP of the current node looks like the following where the matrix includes both the original constraints and the added cuts.

$$\min\{c^T x : Ax = b, \mathcal{L} \leq x \leq U\}.$$

Let J be the set of columns. As usual, let the optimal simplex tableau be rearranged as $A = (A_B, A_N)$. Let $L = \{j \in J : x_j = \mathcal{L}_j\}, U = \{j \in J : x_j = \mathcal{U}_j\}$ be the index sets of non-basic variables at their lower and upper bounds. Let $\bar{A} = A_B^{-1}A, \bar{b} = A_B^{-1}b$. The last simplex tableau implies

$$x_B + \bar{A}_L(x_L - \mathcal{L}_L) - \bar{A}_U(\mathcal{U}_U - x_U) = \bar{b} - \bar{A}_L\mathcal{L}_L - \bar{A}_U\mathcal{U}_U,$$

Similar to the root node cuts, the cuts here are also generated by rounding down the ith row of the above system of equations, provided the corresponding basic variable is fractional.

2.6.4 Modification to Subproblem

We have seen in section 2.1.3, that if we add a constraint in the master problem, we have to make sure that when a new column is generated by the subproblem, the corresponding entry in the new column have the same value as it would have if the column were already present in the master problem. Here, we are fortunate that the coefficient of the cuts are nicely related only with the entries in the same column above it. Thus, it is enough to make sure that the same relation is satisfied in the new column. The subproblem takes the following form:

$$\max \quad \sum_{i=1}^{m} \pi_i a_i + \sum_{r=1}^{\mu} \pi_{m+r} a_{m+r} \tag{2.55}$$

$$\text{s.t.} \quad \sum_{i=1}^{m} w_i a_i \leq W \tag{2.56}$$

$$a_{m+r} = \psi^r(a) = F(\sum_{i=1}^{m} u_i^r a_i + \sum_{t=1}^{r-1} u_{m+t}^r \psi^t(a)) \qquad r = 1, \ldots, \mu \tag{2.57}$$

$$a_i \in \{0, \ldots, a_i^{\max}\} \qquad \forall i = 1, \ldots, m \tag{2.58}$$

$$a_i \in \mathbb{N} \qquad \forall i = m+1, \ldots, m+\mu \tag{2.59}$$

where $\pi \in \mathbb{R}^{m+\mu}$ is an optimal dual solution to the master problem.

However, we do not need the extra variables because we can directly put the right hand

side of the equality constraints in the objective. Thus the subproblem takes the form:

$$\max \quad \sum_{i=1}^{m} \pi_i a_i + \sum_{r=1}^{\mu} \pi_{m+r} \psi^r(a) \tag{2.60}$$

$$\text{s.t.} \quad \sum_{i=1}^{m} w_i a_i \leq W \tag{2.61}$$

$$a_i \in \{0, \ldots, a_i^{\max}\} \qquad \forall i = 1, \ldots, m \tag{2.62}$$

The issue of forbidden patterns is also present here. This is because branch-and-price constraints are also added to the master. Thus a forbidden set is also added with the subproblem formulation.

2.6.5 Solution to Subproblem

Let us assume that there is no forbidden pattern. Then the subproblem looks like a knapsack problem with a modified objective function

$$\bar{c}(a) = \sum_{i=1}^{m} \pi_i a_i + \sum_{r=1}^{\mu} \pi_{m+r} \psi^r(a). \tag{2.63}$$

Let us think about the issues, if we have to use an algorithm like HS1 (see Algorithm 1). We can immediately see that we can not order the items on the basis of π_i/w_i because there is some part of the objective hidden in the expression $\sum_{r=1}^{\mu} \pi_{m+r} \psi^r(a)$. It is difficult to separate out the actual objective of a_i.

Even if we could separate out the objective by expanding the recursively defined $\psi^r(a)$ and regrouping the necessary parts, we can not make an ordering because of the non-linear function F used for generating cuts. This also makes the bounding scheme to fail.

In the next subsections we will see how these two issues are taken care of.

Item Order in Enumeration

Belov and Scheithauer [2006] used the following scheme. The objective function is linearly approximated by omitting the non-linear function. Thus, approximate cut coefficients are defined recursively as

$$\tilde{\psi}_j^r = \tilde{\psi}^r(a^j) = \sum_{i=1}^{m} u_i^r a_{ij} + \sum_{t=1}^{r-1} u_{m+t}^r \tilde{\psi}^t(a^j), \quad r = 1, \ldots, \mu, \tag{2.64}$$

It can be noted that now the expression does not depend on the function F. This also

has linearity because $\tilde{\psi}^r(a) = \sum_{i=1}^{m} a_i \tilde{\psi}^r(e_i)$ for all r, where e_i is the ith unit vector. The approximate objective function is then

$$\tilde{c}(a) = \sum_{i=1}^{m} \pi_i a_i + \sum_{r=1}^{\mu} \pi_r \tilde{\psi}^r(a) = \sum_{i=1}^{m} \tilde{\pi}_i a_i \tag{2.65}$$

with $\tilde{\pi}_i = \pi_i + \sum_{r=1}^{\mu} \pi_{m+r} \tilde{\psi}^r(e^i)$.

With this new objective, the enumeration is done by arranging the items in the order $\tilde{\pi}_1/l_1 \geq \ldots \geq \tilde{\pi}_m/l_m$.

The Upper Bound

In order to avoid full enumeration, after we have filled some entries in the new column, we need an upper bound for the objective value $\bar{c}(a)$ irrespective of how the rest of the entries are filled. If the upper bound is less than the best solution found so far, we can readily skip filling rest of the entries. The proposition 2.6.3 gives such a bound.

Before going to proposition 2.6.3, we prove that the approximation in the objective is not bad. This is done using the proposition 2.6.2. Again, before that, we define three terms $\tilde{\alpha}(x)$, $\bar{\alpha}_r$ and $\underline{\alpha}_r$ recursively as follows:

$$\tilde{\alpha}_r(x) = \begin{cases} x\bar{\alpha}_r, & x \geq 0, \\ x\underline{\alpha}_r, & x < 0 \end{cases}$$

$$\bar{\alpha}_r = \begin{cases} \sum_{t=1}^{r-1} \tilde{\alpha}_t(u^r_{m+t}), & r > 1, \\ 0, & r = 1 \end{cases}$$

$$\underline{\alpha}_r = \begin{cases} -\alpha_r - \sum_{t=1}^{r-1} \tilde{\alpha}_t(-u^r_{m+t}), & r > 1, \\ -\alpha_1, & r = 1 \end{cases}$$

where α_k denotes the parameter used for generating cut k using the generic super-additive, non-decreasing function $F_{\alpha_k}()$. It can be noted that

$$u^T a - \alpha_k \leq F_{\alpha_k}(u^T a) \leq u^T a. \tag{2.66}$$

In particular, when the function is just the rounding down function, $u^T a - 1 \leq \lfloor u^T a \rfloor \leq u^T a$. Before proceeding to the proposition 2.6.2, we prove a result on the three expressions we just defined.

Proposition 2.6.1 *The quantities $\bar{\alpha}_r$ and $\underline{\alpha}_r$ are an upper bound and a lower bound respectively, on the error incurred by approximating rth cut-coefficient $\psi^r(a)$ by $\tilde{\psi}^r(a)$. That is, $\underline{\alpha}_r \leq \psi^r(a) - \tilde{\psi}^r(a) \leq \bar{\alpha}_r$.*

Proof We prove by induction. The relationship is trivially true for $k = 1$. Let us assume

that it is true for $r \leq k-1$. Then,

$$\begin{aligned}
\psi^k(a) &\stackrel{\text{def}}{=} F_{\alpha_k}\left(\sum_{i=1}^{m} u_i^k a_i + \sum_{t=1}^{k-1} u_{m+t}^k \psi^t(a)\right) \\
&\leq \sum_{i=1}^{m} u_i^k a_i + \sum_{t=1}^{k-1} u_{m+t}^k \psi^t(a) \quad \text{using (2.66)} \\
&= \sum_{i=1}^{m} u_i^k a_i + \sum_{t=1}^{k-1} u_{m+t}^k \tilde{\psi}^t(a) + \sum_{t=1}^{k-1} u_{m+t}^k \psi^t(a) - \sum_{t=1}^{k-1} u_{m+t}^k \tilde{\psi}^t(a) \\
&= \tilde{\psi}^k(a) + \sum_{t=1}^{k-1} u_{m+t}^k (\psi^t(a) - \tilde{\psi}^t(a)) \\
&= \tilde{\psi}^k(a) + \sum_{t=1}^{k-1} \begin{cases} u_{m+t}^k (\psi^t(a) - \tilde{\psi}^t(a)), & \text{if } u_{m+t}^k \geq 0, \\ -u_{m+t}^k (\tilde{\psi}^t(a) - \psi^t(a)), & \text{otherwise} \end{cases} \\
&\leq \tilde{\psi}^k(a) + \sum_{t=1}^{k-1} \begin{cases} u_{m+t}^k (\bar{\alpha}_t), & \text{if } u_{m+t}^k \geq 0, \\ -u_{m+t}^k (-\underline{\alpha}_t), & \text{otherwise} \end{cases} \quad \text{by induction hypothesis} \\
&= \tilde{\psi}^k(a) + \sum_{t=1}^{k-1} \begin{cases} u_{m+t}^k \bar{\alpha}_t, & \text{if } u_{m+t}^k \geq 0, \\ u_{m+t}^k \underline{\alpha}_t, & \text{otherwise} \end{cases} \\
&= \tilde{\psi}^k(a) + \sum_{t=1}^{k-1} \tilde{\alpha}_t(u_{m+t}^k) \quad \text{using definition of } \tilde{\alpha}_k(x) \\
&= \tilde{\psi}^k(a) + \bar{\alpha}_k \quad \text{using definition of } \bar{\alpha}_k
\end{aligned}$$

This implies $\psi^k(a) - \tilde{\psi}^k(a) \leq \tilde{\alpha}_k$. Similarly,

$$\begin{aligned}
\psi_k(a) &\stackrel{\text{def}}{=} F_{\alpha_k}\left(\sum_{i=1}^{m} u_i^k a_i + \sum_{t<k} u_{m+t}^k \psi_t(a)\right) \\
&\geq -\alpha_k + \sum_{i=1}^{m} u_i^k a_i + \sum_{t=1}^{k-1} u_{m+t}^k \psi^t(a) \quad \text{using (2.66)} \\
&= -\alpha_k + \sum_{i=1}^{m} u_i^k a_i + \sum_{t=1}^{k-1} u_{m+t}^k \tilde{\psi}^t(a) + \sum_{t=1}^{k-1} u_{m+t}^k \psi^t(a) - \sum_{t=1}^{k-1} u_{m+t}^k \tilde{\psi}^t(a) \\
&= -\alpha_k + \tilde{\psi}^k(a) - \sum_{t=1}^{k-1} -u_{m+t}^k (\psi^t(a) - \tilde{\psi}^t(a)) \\
&= -\alpha_k + \tilde{\psi}^k(a) - \sum_{t=1}^{k-1} \begin{cases} -u_{m+t}^k (\psi^t(a) - \tilde{\psi}^t(a)), & \text{if } -u_{m+t}^k \geq 0, \\ u_{m+t}^k (\tilde{\psi}^t(a) - \psi^t(a)), & \text{otherwise} \end{cases}
\end{aligned}$$

$$
\begin{aligned}
&\geq -\alpha_k + \tilde{\psi}^k(a) - \sum_{t=1}^{k-1} \begin{cases} -u^k_{m+t}(\bar{\alpha}_t), & \text{if } -u^k_{m+t} \geq 0, \\ u^k_{m+t}(-\underline{\alpha}_t), & \text{otherwise} \end{cases} \quad \text{by induction hypothesis} \\
&= -\alpha_k + \tilde{\psi}^k(a) - \sum_{t=1}^{k-1} \begin{cases} -u^k_{m+t}\bar{\alpha}_t, & \text{if } -u^k_{m+t} \geq 0, \\ -u^k_{m+t}\underline{\alpha}_t, & \text{otherwise} \end{cases} \\
&= \tilde{\psi}^k(a) - \alpha_k - \sum_{t=1}^{k-1} \tilde{\alpha}_t(-u^k_{m+t}) \quad \text{using definition of } \tilde{\alpha}_k(x) \\
&= \tilde{\psi}^k(a) + \underline{\alpha}_k \quad \text{using definition of } \underline{\alpha}_k
\end{aligned}
$$

Thus we proved by induction that for all $r = 1, \ldots, \mu$, $\underline{\alpha}_r \leq \psi^r(a) - \tilde{\psi}^r(a) \leq \bar{\alpha}_r$. ∎

Now we state the proposition giving an upper bound on the approximation error.

Proposition 2.6.2 *The objective function $\bar{c}(a)$ of (2.63) is bounded from above as*

$$\bar{c}(a) \leq \tilde{c}(a) + \sum_{r=1}^{\mu} \tilde{\alpha}_r(\pi_{m+r}) \tag{2.67}$$

for any pattern a.

Proof

$$
\begin{aligned}
\bar{c}(a) - \tilde{c}(a) &= \left(\sum_{i=1}^{m} \pi_i a_i + \sum_{r=1}^{\mu} \pi_{m+r}\psi^r(a) \right) - \left(\sum_{i=1}^{m} \pi_i a_i + \sum_{r=1}^{\mu} \pi_{m+r}\tilde{\psi}^r(a) \right) \\
&= \sum_{r=1}^{\mu} \pi_{m+r}\psi^r(a) - \sum_{r=1}^{\mu} \pi_{m+r}\tilde{\psi}^r(a) \\
&= \sum_{r=1}^{\mu} \pi_{m+r}(\psi^r(a) - \tilde{\psi}^r(a)) \\
&= \sum_{r=1}^{\mu} \begin{cases} \pi_{m+r}(\psi^r(a) - \tilde{\psi}^r(a)), & \text{if } \pi_{m+r} \geq 0, \\ -\pi_{m+r}(\tilde{\psi}^r(a) - \psi^r(a)), & \text{otherwise} \end{cases} \\
&\leq \sum_{r=1}^{\mu} \begin{cases} \pi_{m+r}(\bar{\alpha}_r), & \text{if } \pi_{m+r} \geq 0, \\ -\pi_{m+r}(-\underline{\alpha}_r), & \text{otherwise} \end{cases} \\
&= \sum_{r=1}^{\mu} \begin{cases} \pi_{m+r}\bar{\alpha}_r, & \text{if } \pi_{m+r} \geq 0, \\ \pi_{m+r}\underline{\alpha}_r, & \text{otherwise} \end{cases} \\
&= \sum_{r=1}^{\mu} \tilde{\alpha}(\pi_{m+r})
\end{aligned}
$$

That implies $\bar{c}(a) \leq \tilde{c}(a) + \sum_{r=1}^{\mu} \tilde{\alpha}(\pi_{m+r})$. ∎

It can be noted that the approximation errors are constants and hence can be calculated before enumeration.

Now we state the proposition 2.6.3 which gives us the required upper bound.

Proposition 2.6.3 *Let a partial solution $\hat{a} = (\hat{a}_1, \ldots, \hat{a}_s, 0, \ldots, 0)$ be given. Then an upper bound for $\bar{c}()$ on all the patterns built by adding some items of types $i = s+1, \ldots, m$ to $\hat{a}$ is given as follows:*

$$U = \delta\tilde{c} + \tilde{c}(\hat{a}) + \sum_r \tilde{\alpha}_r(\pi_{m+r}), \tag{2.68}$$

where $\delta\tilde{c}$ is the solution to the residual problem (i.e., the problem of filling a knapsack of capacity $W - \sum_{i=1}^{s} w_i\hat{a}_i$ with items $s+1, \ldots, m$ having profits $\tilde{\pi}_{s+1}, \ldots, \tilde{\pi}_m$).

Proof The first two terms give the optimal solution of the problem with approximate objective. Adding the approximation error to it gives the required upper bound (using Proposition 2.6.3). ∎

As we did in $HS1$ (section 2.2), an upper bound on $\delta\tilde{c}$ can be obtained by using LP relaxation of the residual problem.

Enumeration Procedure

With all the result from the previous subsection, we are now at a position to state the enumeration procedure. It works as follows. Suppose, the items $1, \ldots, s$ are already fixed. Now the value of the upper bound as given in (2.6.3) on it is calculated. If it is less than the current best solution, this partial solution is fathomed and we backtrack. On the other hand, if it is greater than the best solution, the original objective function is calculated and checked. If it is still a better solution, it is taken to be the new best.

Now we take care of the issue of forbidden set. As we have done in HS2 (see section 2.2), before updating the current best solution with a new one, we make sure that it is not already present in the forbidden set.

2.6.6 Concluding Notes on the Cutting Plane Algorithm

It can be noted that starting from the root node, just by repeatedly solving master problem using column generation and adding cutting planes, will eventually lead to an integer solution. One may ask – why do we need branching as in branch-and-price then? The experimental result shows that with the initial cutting planes the LP lower bound improves faster. However, it takes many more iterations to reach the final integer solution.

For this, Belov and Scheithauer [2006] used the following strategy. They generated only some limited number of cutting planes to the LP relaxation (10 for the root node and 3 for the internal nodes). If an integer solution is not reached yet, branching is applied. Thus the cutting plane was mainly used to get tighter lower bounds. Since the LP lower bound is already tight for the IRUP instances, it helped only with the non-IRUP instances.

2.7 Conclusions of Literature Survey

We studied the details of the implementations of the different subtasks of the generic branch-and-price algorithm in Degraeve and Schrage [1999], Degraeve and Peeters [2003], Vance et al. [1994], Vance [1998], and Vanderbeck [1999]. Also studied a branch-price-cut algorithm as given in Belov and Scheithauer [2006]. Here we mention our findings.

We studied four different ways of branching. However, it can be noted that the scheme of branching based on binary columns (2.1.2) is a special case of the scheme using a set of columns with IBCs (section 2.1.3). This is evident from the similarity of the proofs of the proposition 2.1.2 and the lemma 2.1.4. The scheme based on a set of columns using ILBCs 2.1.4, though by design, is a special case of the scheme using a set of columns with IBC, turns out to be similar to the branching on a single fractional variable.

Thus we have two main approaches of branching – (i) using a single variable and (ii) using a set of columns with specific properties. The later one seems to be better because it divides the solution space almost equally. However, the former one is popular because of its well known formulation of the subproblem. Here, it is possible to solve the subproblem using some specialized algorithm unlike the other case where the subproblem is formulated as a general integer programming problem.

The hybrid node selection scheme of Belov and Scheithauer [2006] seems better as it balances performance in instances of both types, IRUP and Non-IRUP. The SVC heuristic seems to be the best among the existing approaches, as far as heuristic solutions are concerned. Almost all the algorithms use the same scheme of solving the problem of slow convergence of the column generation process. The use of cuts as in Belov and Scheithauer [2006] proved to be useful in solving the Non-IRUP instances.

Thus, if we have to come up with a better branch-and-price based algorithm, it should have following properties:

1. It solves the LP relaxation at each node efficiently. Any trickier formulation which can accelerate the finding of the optimal set of columns will improve the overall performance of the algorithm.

2. It solves the subproblem faster. The branching scheme based on a single variable is prevalent in the literature. Hence, we should have a better solver for the BKPFS form of the subproblem.

3. It obtains better and quick heuristic solutions. We need to come up with a better approach to obtain heuristic solutions not only quickly but also with better quality.

4. It implements tighter LP relaxation, at least, for the non-IRUP instances. Cutting plane like ideas help in Non-IRUP instances where the LP relaxation does not give a tight lower bound.

2.8 Summary

We discussed several implementations of the generic branch-and-price algorithms and a branch-cut-price algorithm. We concluded the literature survey by identifying the tasks that we should work on to improve the performance of the branch-and-price algorithms.

Chapter 3

Our Contribution

In this chapter, we describe our contribution on improving the branch-and-price algorithms. The first thing we did is to implement a working version of the generic branch-and-price algorithm. We implemented the algorithm using the BCP framework provided by COIN-OR. We describe the implementation in section 3.1. The solution to subproblem is discussed in section 3.2.

In section 2.7 we mentioned that one of the tasks identified is to accelerate the column generation process. In section 3.3 we describe a scheme of modifying the LP relaxation to accelerate the column generation. We experimented with a scheme to obtain the heuristic solution faster. We describe that in section 3.4.

3.1 COIN-OR Based Implementation

We implemented a version of the basic branch-and-price algorithm using the BCP framework provided by COIN-OR, `http://www.coin-or.org`. We used this open source framework so that we do not have to rewrite code afresh for many of the subtasks.

We implemented the simple branching scheme using a single variable. The subproblem is solved using a dynamic programming algorithm described in section 3.2. The heuristic solutions are obtained using SVC method.

3.2 Dynamic Programming Solution to Subproblem

We have seen that if the conventional branching with a single variable is used, the subproblem takes the form of a bounded knapsack problem constrained with a list of forbidden patterns.

We used the following dynamic programming solution to the BKPFS. We introduce few

notations before explaining the algorithm. $\phi_i(a)$ is the vector comprised of the first i components of the vector a. $\phi_i(S)$ is the set comprising of the first i components of vectors in S, i.e., $\phi_i(S) = \{\phi_i(a) \mid a \in S\}$. $\varphi_i(a,k)$ is the vector comprised of the first i components of the vector a such that ith component is k, i.e., $\varphi_i(a,k) = (\phi_i(a) \mid a_i = k)$. $\varphi_i(S,k) = \{\varphi_i(a,k) : a \in S\}$.

Let $f(i,\hat{W},\hat{S})$, $i \in \{0,1,\ldots,m\}$, $\hat{W} \in \{0,1,\ldots,W\}$, $\hat{S} \subseteq \phi_i(S)$, be an optimal solution to the following subproblem of the BKPFS.

$$\max \quad \sum_{j=1}^{i} \pi_j a_j \tag{3.1}$$

$$\text{s.t.} \quad \sum_{j=1}^{i} w_j a_j \leq \hat{W} \tag{3.2}$$

$$a_j \in \{0,1,\ldots,b_j\} \qquad \forall j \in \{1,2,\ldots,i\} \tag{3.3}$$

$$(a_1,a_2,\ldots,a_i) \notin \hat{S} \tag{3.4}$$

We use the following recursion for the solution of BKPFS:

$$f(i,\hat{W},\hat{S}) = \max \begin{cases} f(i-1,\hat{W},\varphi_i(\hat{S},0)) & \text{if } \hat{W} \geq 0 \\ f(i-1,\hat{W}-w_i,\varphi_i(\hat{S},1)) & \text{if } \hat{W}-w_i \geq 0 \\ \vdots & \\ f(i-1,\hat{W}-b_i w_i,\varphi_i(\hat{S},b_i)) & \text{if } \hat{W}-b_i w_i \geq 0 \end{cases} \tag{3.5}$$

with the boundary values $f(0,\hat{W},\varnothing) = 0,\ f(0,\hat{W},\hat{S} \neq \varnothing) = -\infty$.

Worstcase Time Complexity

It can be seen that the subproblem has three input parameters which vary. The parameters are i, $\hat{W}$ and $\hat{S}$. Let $b_{max} = \max\{b_i\}$. A quick calculation should reveal that the size of the dynamic programming table is = O(no of possible values of i $\times$ no of possible values of W $\times$ no of possible values of S) = $O(m \times W \times b_{max}^m)$ since the number of configurations, and hence possibly the forbidden set, of m items is b_{max}^m.

However, a deeper look into the operation on the table will reveal that only a few entries of the dynamic programming table are in fact accessed by the algorithm. For the particular value of i and $\hat{W}$ there can be at most b_i possible values of $\hat{S}$, this is because the corresponding item could be present b_i times in the forbidden list. The maximum number of entries in the table that are accessed by the algorithm is $O(b_{max}mW)$. Again for filling each of these entries takes $O(|S|)$ time. Hence, the time complexity = $O(b_{max}mW|S|)$.

The time complexity of the branch-and-bound algorithm, HS2 (see 2.2) is $O(b_{max}^m + m|S|)$.

This is because, in the worst case, the branch-and-bound tree can be the complete b_{max}-ary tree with height m and since at each level the list of $|S|$ forbidden columns is distributed among the nodes, the time required summed up for a level is $|S|$.

Hence, we see that if $W < b_{max}^n$, in the worst case, dynamic programming solution works better. On the other case, branch-and-bound algorithm works better.

3.3 Accelerating Column Generation

We mentioned in the section on our learning from literature survey (see 2.7) that one way to improve the performance is to accelerate the column generation process while solving the LP relaxation at any node in the branch and bound tree. For solving the LP relaxation, initially, we take a very restricted or constrained form of the master problem with only few columns enough to ensure the feasibility and gradually relax the formulation by adding new columns. This is continued until we get an optimal solution. Any trickier formulation to accelerate the finding of the optimal set of columns should make this relaxation process faster.

To find such a formulation that relaxes the master problem, we thought of allowing the demand of an item of smaller width be met by an item of larger width. Mathematically, suppose we need to cut b_1 number of items of width w_1 and b_2 number of items of width w_2 where $w_1 \geq w_2$. We can relax this requirement by saying that we require b_1 number of items of width w_1 and together $b_1 + b_2$ number of items of width either w_2 or w_1. This is reasonable because if in the solution the demand of w_2 is met by w_1, we can easily replace w_1 by w_2 in the corresponding cutting pattern though the cutting may not be optimal.

In the following section, we state the formulation formally. This scheme is named as 'GoodPrice' as we will see in section 3.3.2 that in the optimal solution to the dual of the modified formulation, the variables are also in the same order as the widths of the items.

3.3.1 Formulation

Before stating the new formulation, let us restate the constraints of the original formulation.

$$a_{1*}x \geq b_1 \tag{3.6}$$
$$a_{2*}x \geq b_2 \tag{3.7}$$
$$\dots \tag{3.8}$$
$$a_{m*}x \geq b_m \tag{3.9}$$

where a_{i*} is the i^{th} row of the matrix A.

If we apply the relaxation as described above, the constraints take the form

$$a_{1*}x' \geq b_1 \tag{3.10}$$
$$(a_{1*} + a_{2*})x' \geq b_1 + b_2 \tag{3.11}$$
$$\dots \tag{3.12}$$
$$(a_{1*} + a_{2*} + \dots + a_{m*})x' \geq b_1 + b_2 + \dots + b_m \tag{3.13}$$

note that solution to the relaxed system may not be a solution to the original system and that is why new variable x' is used.

3.3.2 Proof of Correctness

It needs to prove that with the relaxed primal, the solution obtained at the end of column generation process, the objective would be same as that objective that would have been obtained if the relaxation was not used. We prove the correctness of the 'GoodPrice' using two lemmas.

Lemma 3.3.1 *The 'GoodPrice' relaxation is equivalent to adding the constraint* $\pi_1 \geq \pi_2 \geq \dots \geq \pi_m$ *in its dual.*

Proof The relaxed formulation is equivalent to pre-multiplying both sides by a lower triangular matrix L with all non-zero elements 1 i.e.

$$L = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 1 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 1 & 1 & \dots & 1 \end{bmatrix} \tag{3.14}$$

Thus, the related master LP is

$$\min \quad \mathbf{1}^T x' \tag{3.15}$$
$$\text{s.t.} \quad LAx' \geq Lb \tag{3.16}$$
$$x' \geq 0 \tag{3.17}$$

The corresponding dual is

$$\max \quad b^T L^T \pi' \tag{3.18}$$
$$\text{s.t.} \quad A^T L^T \pi' \leq \mathbf{1} \tag{3.19}$$
$$\pi' \geq 0 \tag{3.20}$$

If we substitute $L^T\pi'$ by π, i.e., $\pi' = (L^T)^{-1}\pi$, we have

$$\max \quad b^T\pi \tag{3.21}$$
$$\text{s.t.} \quad A^T\pi \leq \mathbf{1} \tag{3.22}$$
$$(L^T)^{-1}\pi \geq 0 \tag{3.23}$$

However, $(L^T)^{-1}$ is given by

$$(L^T)^{-1} = \begin{bmatrix} 1 & -1 & 0 & \dots & 0 \\ 0 & 1 & -1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & -1 \\ 0 & 0 & \dots & 0 & 1 \end{bmatrix} \tag{3.24}$$

The extra constraint (3.23) implies

$$\pi_1 \geq \pi_2 \geq \cdots \geq \pi_m \tag{3.25}$$

∎

Lemma 3.3.2 *The dual solution to the original formulation (1.1..1.3) implicitly satisfies the constraints* (3.25)*.*

Proof We prove by contradiction. Suppose that the solution to the original formulation (1.1..1.3) has two dual variables π_k and π_l such that $\pi_k < \pi_l$ and $w_k \geq w_l$. Suppose, in the corresponding primal solution, $x_j > 0$, where the pattern j contains the item k. Let the column for the pattern j be given by $[a_1 \quad \dots \quad a_k \quad \dots \quad a_l \quad \dots \quad a_m]^T$. Using complementary slackness, in the dual, the constraint corresponding to pattern j is tight. That means,

$$a_1\pi_1 + \ldots + a_k\pi_k + \ldots + a_l\pi_l \ldots + a_m\pi_m = 1 \tag{3.26}$$

However, since, $w_k \geq w_l$ if we replace each instance of item k in pattern j by an instance of item l, we get a new valid pattern. The column for the new pattern is given by $[a_1 \quad \dots \quad 0 \quad \dots \quad a_k + a_l \quad \dots \quad a_m]^T$. The left hand of the corresponding constraint in the dual is

$$a_1\pi_1 + \ldots + 0\pi_k + \ldots + (a_k + a_l)\pi_l \ldots + a_m\pi_m \tag{3.27}$$
$$> a_1\pi_1 + \ldots + a_k\pi_k + \ldots + a_l\pi_l \ldots + a_m\pi_m \qquad \pi_l > \pi_k \tag{3.28}$$
$$= 1 \tag{3.29}$$

That means, the constraint in the dual is violated. That is not possible. ∎

The two lemmas 3.3.1 and 3.3.2 imply that with the extra constraint (3.25), we have, in fact, added no extra constraint to the formulation. Hence, the solution obtained after adding the extra constraints is same as that would have been obtained without adding it.

3.4 A Quick Heuristic Approach

The update to pseudo-cost in the sequential value correction method (section 2.4.2) is adhoc in the sense it gives some arbitrary weight to the old and new values. We wanted to come up with a more theoretically sound update scheme. Before describing the heuristic, we give the following two propositions.

Proposition 3.4.1 *Suppose b be the initial demand and x^*, y^* be an optimal primal-dual solution pair to the LP relaxation of the master problem. Let the rounded down primal solution be $\underline{x}^*$ and b' be the residual demand. Then y^* is also a dual LP optimal solution the residual problem.*

Proof Let $fr(x)$ denote the fractional part of x, i.e. $fr(x) = x - \lfloor x \rfloor$. Then $fr(x^*)$ is a feasible solution to the residual primal LP. This is because $Afr(x^*) = A(x^* - \underline{x}^*) = Ax^* - A\underline{x}^* \geq b - A\underline{x}^* = b'$. Also, y^* is a feasible solution to the dual because the constraints in the dual remain same. Now from LP duality theorem,

$$\begin{aligned} \sum_j x_j^* &= b.y^* \\ \Rightarrow \sum_j \underline{x}_j^* + \sum_j fr(x_j^*) &= b.y^* \\ \Rightarrow \sum_{j, x_j^* > 0} \underline{x}_j^* + \sum_j fr(x_j^*) &= b.y^* \end{aligned}$$

Using complementary slackness, the constraints in the dual corresponding to the non-zero primal variables are tight. That means

$$\begin{aligned} \sum_i y_i^* a_{ij} &= 1 \qquad\qquad \forall j, x_j^* > 0 \\ \text{i.e.,} \quad a_{*j}.y^* &= 1 \end{aligned}$$

Now

$$
\begin{aligned}
b' &= b - \sum_{j,x_j^*>0} \underline{x}_j^* a_{*j} \\
\Rightarrow b &= b' + \sum_{j,x_j^*>0} \underline{x}_j^* a_{*j} \\
\Rightarrow b.y^* &= b'.y^* + (\sum_{j,x_j^*>0} \underline{x}_j^* a_{*j}).y^* \\
\Rightarrow b.y^* &= b'.y^* + \sum_{j,x_j^*>0} \underline{x}_j^* (a_{*j}.y^*) \\
\Rightarrow b.y^* &= b'.y^* + \sum_{j,x_j^*>0} \underline{x}_j^* \\
\Rightarrow b'.y^* &= \sum_j fr(x_j^*)
\end{aligned}
$$

That means $fr(x^*)$ and y^* are a primal-dual solution pair for the residual problem. Hence they must be an optimal pair. ∎

Proposition 3.4.2 *If y^* is used for generating a new column solving the residual problem, the optimal objective of the subproblem is ≤ 1.*

Proof It is true otherwise the column generation would have picked this column before stopping. ∎

More over, if a column of objective 1 is chosen and a new residual problem is formed by subtracting this column from the residual demand, there is a possibility that the optimal LP objective of the new residual problem is 1 less than the optimal LP solution of first residual problem. This implies that if can continue this process, eventually the LP solution gets converted into a integer solution.

Using the above two propositions and the above observation we came up with the following scheme for generating the heuristic solution.

1. Get the rounded down solution and form the residual problem
2. Generate a column selecting at random from the all possible columns with objective 1 using a modified version of the dynamic programming solver for the subproblem
3. If the LP solution of the new residual problem after subtracting the generated column satisfies the relationship mentioned in the above observation, recurse with the new residual problem

4. Otherwise, reject this new column and backtrack using an alternate column of objective 1

3.5 Summary

We discussed about an implementation of the basic branch-and-price algorithm. We also implemented a scheme to accelerate the column generation process. An approach to get heuristic solutions quickly was described.

Chapter 4

Experimental Results

In chapter 3, we discussed about the two ideas we worked on with an aim to improve the performance of the branch-and-price based algorithms. In this chapter, we publish the results of implementing the ideas on our implementation of the generic branch-and-price algorithm.

One idea, named GoodPrice, is to modify the master formulation to accelerate the column generation process so that an optimal LP relaxation is obtained quickly. We show the result of implementing this idea in section 4.1. The results on the other idea to generate quick heuristic solutions are provided in section 4.2.

It should be noted that for the solving the subproblem, we implemented the solver based on the dynamic programming solution of BKPFS only. Hence, we do not show any results for it.

4.1 GoodPrice Results

We measured the performance of the GoodPrice scheme on two sets of CSP examples. The first set of examples were constructed randomly with item widths in the range $1..1,000$ and stock width $10,000$ such that the items fit exactly in two stocks. We call this set as 'duplet'. The second set of example is a subset of the 'hard28' set of one dimensional bin packing problems as given in Belov and Scheithauer [2006]. It can be noted that a bin packing problem is an instance of the cutting stock problem where the items are ordered in small quantities.

4.1.1 Results on 'Duplet' Set

In table 4.1 we show, as a first result, the performance improvement in solving the root node master LP relaxation. It can be seen that GoodPrice based solution took on an

average 54% less number of iterations compared to the simple column generation process. However, the time taken is increased by 9% on an average. Increase in time is due to the fact that the subproblem becomes harder to solve. However, we can anticipate that the gain in using the scheme may fructify in the overall branch and price process where the LP is not often solved to optimality. It can be seen that time to solve the master LP relaxation is almost constant per iteration in both the cases. We will omit this in the subsequent tables.

The improvement over all the nodes in the branch-and-bound tree is shown in table 4.2. As we expected, except in two instances, the scheme generates faster solution. We see that number of columns generated is lesser by 33%. The reduction of this percentage can be explained by the fact that as we go deeper in the tree, the master LP becomes more constrained and the effectiveness of the scheme reduces. On the other hand the time to solve the subproblems also get reduced by around 33%. This increase in the percentage improvement on the subproblem time can be explained by the fact that in the overall solution the column generation is stopped early, if possible, so that the we do not need to solve many 'hard' instances of the subproblem which are generally formed at the end stage of the column generation process. Thus, overall, the time taken is reduced by 33%.

4.1.2 Results on 'Hard28' Set

The results for the 'hard28' set is given in Table 4.3. With our dynamic programming solution to the BKPFS, it was not possible to solve the instances for integer solution. The results correspond to solving the root only. It should be noted that, we have used the Bounded Knapsack Solver (see the future works section 5.2.1) which is applicable only to the root node where the forbidden set is empty. The number of iterations were reduced by 28%. More reduction (44%) on the overall time is because of the faster subproblem solver.

4.2 Quick Heuristic Results

The results on the quick heuristic is shown in Table 4.4. The result shows the quick heuristic applied to 'hard28' set. Because of the limitation of the subproblem solver, we measured the quantities only for getting the heuristic solution for the root node LP. All except the last 5 are IRUP instances. Compared to the SVC approach of finding a heuristic solution directly, our approach assumes that there is a solution of a particular value and tries to find that. For that reason, we executed three sets of experimention. In the first two case, we assumed that there is an IRUP solution. In the third we assumed that there is a MIRUP solution. The difference between the first two is that in the first case we tried with maximum 5 best solutions of objective 1 returned by the modified dynamic

Table 4.1: Experimental result for GoodPrice in solving the root node of 'duplet'

#	Gilmore Gomory Formulation				GoodPrice				% Gain			
	TT	NS	MT	ST	TT	NS	MT	ST	TT	NS	MT	ST
1	35.32	163	0.24	35.08	25.91	67	0.11	25.79	26.64	58.90	54.17	26.48
2	47.74	160	0.28	47.46	36.04	65	0.10	35.94	24.51	59.38	64.29	24.27
3	30.87	144	0.19	30.68	39.84	62	0.08	39.76	-29.06	56.94	57.89	-29.60
4	39.07	119	0.10	38.98	36.41	61	0.06	36.35	6.81	48.74	40.00	6.75
5	26.08	158	0.26	25.82	23.47	70	0.11	23.36	10.01	55.70	57.69	9.53
6	48.52	163	0.29	48.22	60.81	70	0.12	60.69	-25.33	57.06	58.62	-25.86
7	42.06	142	0.19	41.87	49.15	71	0.10	49.06	-16.86	50.00	47.37	-17.17
8	24.27	133	0.16	24.11	44.24	64	0.10	44.14	-82.28	51.88	37.50	-83.08
9	27.21	164	0.30	26.91	23.13	69	0.12	23.01	14.99	57.93	60.00	14.49
10	43.71	161	0.24	43.47	49.88	63	0.10	49.78	-14.12	60.87	58.33	-14.52
Avg									-8.47	55.74	53.59	-8.87

TT Total Time
NS Number of Subproblems
MT Master Time
ST Subproblem Time

Table 4.2: Experimental result for GoodPrice in complete solution of 'duplet'

Test#	Gilmore Gomory Formulation			GoodPrice			% Gain		
	TotTime	NumSubp	SubpTime	TotTime	NumSubp	SubpTime	TotTime	NumSubp	SubpTime
1	154.102	228	132.344	70.152	115	59.368	54.48	49.56	55.14
2	136.669	195	118.063	111.547	167	95.486	18.38	14.36	19.12
3	74.489	128	63.292	52.271	89	44.491	29.83	30.47	29.71
4	88.142	199	75.361	34.222	79	29.226	61.17	60.3	61.22
5	98.206	158	83.205	103.198	158	88.298	-5.08	0	-6.12
6	172.663	232	149.861	107.371	159	92.314	37.81	31.47	38.4
7	135.140	205	116.135	104.355	154	90.146	22.78	24.88	22.38
8	78.149	139	66.108	30.974	55	26.202	60.37	60.43	60.37
9	141.133	213	121.192	165.234	211	143.957	-17.08	0.94	-18.78
10	264.561	326	230.102	82.525	119	71.068	68.81	63.5	69.11
Avg							33.147	33.590	33.054

Table 4.3: Experimental result for GoodPrice in solving root node of 'hard28'

Test#	Gilmore Gomory Formulation			GoodPrice			% Gain		
	TotTime	NumSubp	SubpTime	TotTime	NumSubp	SubpTime	TotTime	NumSubp	SubpTime
BPP13	6.392	803	0.192	4.168	669	0.224	34.794	16.687	-16.669
BPP144	6.772	781	0.240	3.628	518	0.176	46.426	33.675	26.668
BPP178	7.016	830	0.180	3.796	649	0.256	45.895	21.807	-42.223
BPP181	4.716	762	0.196	2.576	525	0.164	45.377	31.102	16.327
BPP195	7.076	777	0.136	4.132	538	0.204	41.605	30.759	-50.000
BPP360	3.700	597	0.116	2.164	459	0.108	41.514	23.116	6.894
BPP40	5.264	729	0.188	2.904	491	0.124	44.833	32.647	34.045
BPP419	7.981	876	0.300	4.332	653	0.272	45.714	25.457	9.334
BPP47	5.252	685	0.096	2.460	531	0.088	53.161	22.482	8.330
BPP485	5.412	783	0.252	2.652	521	0.216	50.998	33.461	14.285
BPP531	4.436	700	0.108	2.632	561	0.096	40.667	19.857	11.104
BPP561	7.957	783	0.176	4.768	537	0.180	40.070	31.418	-2.270
BPP60	3.496	644	0.208	1.948	470	0.180	44.279	27.019	13.461
BPP640	4.548	659	0.116	2.196	458	0.088	51.715	30.501	24.136
BPP645	4.508	680	0.136	2.876	483	0.148	36.202	28.971	-8.824
BPP709	5.988	745	0.216	3.732	544	0.196	37.675	26.980	9.264
BPP716	4.072	677	0.128	1.816	480	0.088	55.403	29.099	31.250
BPP742	3.316	594	0.128	2.004	445	0.132	39.566	25.084	-3.124
BPP766	4.200	692	0.220	1.996	451	0.176	52.476	34.827	20.000
BPP781	9.153	858	0.228	7.080	743	0.228	22.640	13.403	-0.003
BPP785	7.152	849	0.256	4.452	613	0.228	37.752	27.797	10.936
BPP814	5.228	672	0.104	1.888	397	0.120	63.887	40.923	-15.392
BPP832	3.272	618	0.184	2.244	457	0.164	31.418	26.052	10.871
BPP900	7.396	869	0.292	3.876	598	0.252	47.593	31.185	13.698
BPP119	6.824	838	0.228	3.620	600	0.232	46.952	28.401	-1.755
BPP14	3.212	592	0.204	2.112	454	0.164	34.247	23.311	19.606
BPP175	6.908	857	0.244	3.664	608	0.164	46.960	29.055	32.788
BPP359	4.608	695	0.144	2.380	526	0.176	48.351	24.317	-22.221
BPP716	4.056	677	0.152	1.812	480	0.128	55.325	29.099	15.790
Avg							44.258	27.534	5.735

programming solver for the subproblem. In the second case, we tried with maximum 10 best solutions.

It can be seen from the table that assuming IRUP, we could not find any solution. However, assuming MIRUP, we got solutions for all except 1. It can be also seen that the SVC also got the same solution. The quick heuristic could find the solution in lesser number of iterations. By iterations, we mean the number of extra patterens generated in the heuristic procedure. However, it can be mentioned that we could have also altered the SVC to look for MIRUP solution. Thus we can conclude that the quick heuristic solution is no better than the SVC.

4.3 Summary

We showed the results for the two ideas we worked on. The GoodPrice scheme is shown to improve performance by 33%. The quick heuristic approach does not improve solution compared to the SVC method.

Table 4.4: Experimental result for quick heuristic

Test#	LP Value	IP Value	SVC		IRUP, Best 5		IRUP, Best 10		MIRUP, Best 5	
			HeuSol	Iters	HeuSol	Iters	HeuSol	Iters	HeuSol	Iters
BPP13	69.999	67	68	800	no	13	no	25	68	1697
BPP144	73.000	73	74	700	no	1	no	1	74	109
BPP178	79.995	80	81	500	no	12	no	15	81	37
BPP181	71.999	72	73	600	no	3	no	16	73	49
BPP195	63.996	64	65	1000	no	10	no	19	65	368
BPP360	62.000	62	63	800	no	1	no	1	63	3241
BPP40	58.999	59	60	500	no	4	no	2	60	96
BPP419	79.999	80	81	500	no	1	no	3	81	123
BPP47	71.000	71	72	800	no	1	no	1	72	2580
BPP485	70.997	71	72	500	no	6	no	8	72	187
BPP531	83.000	83	84	500	no	1	no	1	84	3330
BPP561	71.996	72	73	1101	no	8	no	15	73	301
BPP60	62.998	63	64	500	no	4	no	8	64	50
BPP640	74.000	74	75	400	no	1	no	1	75	149
BPP645	57.999	58	59	700	no	2	no	1	59	50
BPP709	67.000	67	68	700	no	1	no	1	68	188
BPP742	64.000	64	65	700	no	1	no	1	65	255
BPP766	61.999	62	63	600	no	3	no	5	63	48
BPP781	70.999	71	72	600	no	1	no	5	no	11
BPP785	67.994	68	69	800	no	19	no	47	69	103
BPP832	59.998	60	61	600	no	6	no	9	61	97
BPP119	76.000	77	77	800					77	90
BPP14	60.998	62	62	701					62	108
BPP175	83.000	83	84	600					84	1198
BPP359	74.9983	76	76	600					76	151
BPP716	75.0000	76	76	400					76	41

Chapter 5

Conclusions and Future Works

In this chapter, we conclude our report. In section 5.1, we give a summary of the tasks we were able to achieve. The future works that can be undertaken are described in section 5.2.

5.1 Conclusions

We studied some of the branch-and-price based algorithms existing in literature for solving the one dimensional cutting stock problem. We came to the conclusion that a better branch-and-price based algorithm, should have following properties:

1. It solves the LP relaxation at each node efficiently.
2. It solves the subproblem faster.
3. It obtains better and quick heuristic solutions.
4. It implements tighter LP relaxation, at least, for the non-IRUP instances.

With this finding, we worked on the following

1. An implementation of the generic branch-and-price algorithm in which the subproblem was solved by a dynamic programming algorithm.
2. Worked on a scheme, named GoodPrice, to accelerate the solution of the LP relaxation using column generation method
3. A quick heuristic solution

The GoodPrice scheme was found to improve performance by 33%. However, the quick heuristic approach did not improve solution and performance compared to the SVC method.

5.2 Future Works

Our implementation is a good starting point. However, this basic implementation is not as fast as the implementation existing in the current literature. Hence, though our new technique gives better performance over our base performance, it is not good overall. Thus, a faster base implementation is urgently needed. Also, we have few more ideas which are supposed to improve the overall performance but could not implement. Those need to be implemented. There are few directions where further research may improve the algorithms. In this section we mention all of them.

Depending on the difficulty we classify the future works into three categories. Short term goals are the ones which are urgently needed and can be easily implemented. Medium term goals are the ones which are not totally new but will require more research. Long term goals are on using ideas which are comparatively new and there is a hope that they may improve the performance further.

5.2.1 Short Term Goals

Improvement of the base implementation is the need of the hour. We found out that the slow performance is mainly because of the dynamic programming solution to the subproblem in the form of a Bounded Knapsack Problem with Forbidden Solution. We have to make sure that there are no implementation issues. For that an analysis of the implementation is needed. We have already thought of an improvement on the dynamic programming solution. We tried the implementation of the bounded knapsack solver of Pisinger [2000] for solving the subproblem on the root node where the forbidden set is empty. We found out significant improvement on the performance. It makes the subproblem solver around 800 times faster. The algorithm also solves the problem using dynamic programming but significantly reduces the number of states explored by using dominance relation on the states and an upper bound on the objective over a set of states. We have also figured out that similar dominance relation and bounding is applicable even when there are forbidden sets. We need to implement them.

We also have some reservations on the performance of solvers used in COIN-OR. We did some experimentation and found out that the CPLEX LP solver is around 4 times faster and the CPLEX IP solver is around 40 times faster on the problems generated on the CSP instances we experimented. We should try to setup our implementation to work using the CPLEX solvers. Also, we may have to come up with a branch-and-bound framework specific for our need unlike the generic branch-and-bound framework of COIN-OR.

One of the few things that we wanted to do is to have a clear idea about the exact impact of different tricks or choices used by the implementations existing in the literature on the overall performance. Some of them is easy to figure out. However, some are not. For example, Degraeve and Peeters [2003] with branching on single variables claims to have

performance similar to that of Vanderbeck [1999] which implements branching using a set of columns. From this we can not conclude whether one branching scheme is better than the other. The best thing should be to implement both the approaches and make the comparison. For that, we need to have an experimental setup where we can generate random instances of varying difficulty as followed in many of the existing literature.

5.2.2 Medium Term Goals

We have seen that using cutting planes to tighten the LP relaxation at a node in the branch-and-bound tree is one of the most recent ideas for improvement. We thought about strengthening cutting planes used in Belov and Scheithauer [2006]. We thought that we should first form an integer solution, say v using the current columns. Then add cut a by lower bounding the current solution by v. This is a tight constraint on the restricted master. We also thought of correspondingly modifying the subproblem. However, we found that the integer solution to the restricted master (an instance of the set cover problem) is hard to find.

Our experimentation on 'GoodPrice' and on the quick approach showed that one of the most challenging problem in branch-and-price algorithms is to devise mechanisms so that after we solve the restricted master problem, the dual variables get assigned sensibly, i.e. the pseudo-costs gets assigned sensibly to the various items. Cutting planes also help in reassigning the dual variables. Thus, thinking about this directly might be useful in general for developing cutting plane like ideas.

5.2.3 Long Term Goals

It seems that the solution to the Gilmore-Gomory formulation depends on the structure of the corresponding knapsack problem. Even we can totally omit the master problem and restate the cutting stock problem as finding minimum k such that the demand vector b_i can be expressed as sum of k integer points in the knapsack polytope. The dependence on the knapsack polytope is evident from the following fact stated in Baum and Trotter Jr [1981].

Proposition 5.2.1 (Baum and Trotter Jr [1981]) *A CSP instance satisfies IRUP iff the corresponding knapsack polyhedron has integral decomposition property. A polyhedron $P \subseteq \mathbb{R}^n_+$ has integral decomposition property if for every integer k, any integral vector in the polyhedron $\{kx \mid x \in kP\}$ can be expressed as sum of k integral vector in P.*

Thus, one possible direction of research would be to get a proof of the conjecture on MIRUP of CSP instances so that the insight gained will help in improving the branch-and-price algorithms we described in this report. However, this may require significant effort.

References

S. Baum and L. E. Trotter Jr. Integer Rounding for Polymatroid and Branching Optimization Problems. *SIAM Journal on Algebraic and DiscreteMethods*, 2:416–425, 1981.

G. Belov and G. Scheithauer. A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting. *European journal of operational research*, 171(1):85–106, 2006.

Z. Degraeve and M. Peeters. Optimal Integer Solutions to Industrial Cutting-Stock Problems: Part 2, Benchmark Results. *INFORMS Journal on Computing*, 15(1):58–81, 2003.

Zeger Degraeve and Linus Schrage. Optimal integer solutions to industrial cutting stock problems. *INFORMS J. on Computing*, 11(4):406–419, 1999. ISSN 1526-5528.

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

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

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

E. Horowitz and S. Sahni. Computing Partitions with Applications to the Knapsack Problem. *Journal of the ACM (JACM)*, 21(2):277–292, 1974.

E. Johnson and M. W. Padberg. A note on the knapsack problem with special ordered sets. *Operations Research Letters*, 1:18–22, 1981.

D. Pisinger. A Minimal Algorithm for the Bounded Knapsack Problem. *INFORMS Journal on Computing*, 12(1):75–82, 2000.

D.M. Ryan and BA Foster. An integer programming approach to scheduling. *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling*, pages 269–280, 1981.

P.H. Vance. Branch-and-Price Algorithms for the One-Dimensional Cutting Stock Problem. *Computational Optimization and Applications*, 9(3):211–228, 1998.

P.H. Vance, C. Barnhart, E.L. Johnson, and G.L. Nemhauser. Solving binary cutting stock problems by column generation and branch-and-bound. *Computational Optimization and Applications*, 3(2):111–130, 1994.

F. Vanderbeck. Computational study of a column generation algorithm for bin packing and cutting stock problems. *Mathematical Programming*, 86(3):565–594, 1999.

F. Vanderbeck and LA Wolsey. An exact algorithm for IP column generation. *Operations Research Letters*, 19(4):151–159, 1996.