CUTTING STOCK PROBLEM BASED ON THE LINEAR PROGRAMMING APPROACH

BY

PHILIP DEBRAH (B.Sc. Computer Science)

A thesis submitted to the Department of Mathematics, Kwame Nkrumah

University of Science and Technology, Kumasi

in partial fulfilment of the requirements for the degree of

Master of Science

Industrial Mathematics

INSTITUTE OF DISTANCE LEARNING

OCTOBER, 2011

DECLARATION

I hereby declare that this submission is my own work towards the Master of Science Industrial Mathematics and that, to the best of my knowledge it contains no material previously published by another person or material which has been accepted for award of any other degree of the university except where due acknowledgement has been made in the text.

Philip Debrah, PG3011809	.....	.....
Student's Name & ID	Signature	Date

Certified by:

Mr. K. F.Darkwah	.....	.....
Supervisor's Name	Signature	Date

Certified by:

Mr. K. F.Darkwah	.....	.....
Head of Department's Name	Signature	Date

Certified by:

Prof. I.K Dontwi	.....	.....
Dean of IDL	Signature	Date

ABSTRACT

This thesis considers the application of the Cutting Stock Problem based on the Linear Programming Approach. This is applied in the cutting of paper, glass, steel rod, wood etc. In this thesis, we apply a variant of it, the One-dimensional Cutting Stock Problem, to the cutting of wood in a sawmill. In a sawmill, boards are first cut along their width (rip) into strips, then the obtained strips are cut along their length (strip cut) into cut-pieces with specific length and demand.

The thesis focuses on using simplex algorithm to find optimal cutting patterns. In the simplex algorithm, to determine the entering column (pattern), we solve sub-problem. The sub-problem is of a knapsack type and we solve it using dynamic programming. We develop a computer program based on the above approach to generate optimal cutting patterns.

Keywords: One-dimensional Cutting Stock Problem, Linear programming, knapsack problem, simplex algorithm, dynamic programming,

TABLE OF CONTENT

Content	*Page*
Declaration	ii
Abstract	iii
Table of contents.....	iv
List of tables	viii
List of figures.....	ix
Dedication.....	x
Acknowledgement	xi
CHAPTER ONE: INTRODUCTION.....	1
1.1 The Cutting Stock Problem.....	1
1.2 Background to Rogersco Sawmill Limited.....	2
1.3 Problem Statement.....	3
1.4 Objective(s) of the Study.....	3
1.5 Methodology.....	4
1.6 Justification	4
1.7 Thesis Organization.....	6
CHAPTER 2: LITERATURE REVIEW	
2.1 A Review of the Cutting Stock Problem.....	7
2.3 Historical background of the Cutting Stock Problem.....	8

CHAPTER 3: METHODOLOGY

3.1 General Overview of Cutting Stock Problem……..……. 16

3.2 Classification of the Cutting Stock Problem……..…… 17

3.2 .1 One dimensional Cutting Stock Problem……..…… 17

3.2 .2 Two dimensional Cutting Stock Problem……..…… 18

3.2 .3 Three dimensional Cutting Stock Problem……..…… 21

3.3 Strip Board Cutting as a One Dimensional Cutting Problem…… 21

3.3.1 General Formulation of The Cutting Stock Problem……..……. 21

3.3.2Formulation of the auxiliary Knapsack Problem……. 24

3.4 Methods of Solution For the Cutting Stock Problem…..…… 28

3.4.1 Linear Programming…….…. 28

3.4.2General Formulation of a Linear Program…… 29

3.4.3 The Simplex Method…… ..…… 32

3.4.4 Column Generation……..…… 36

3.5 The Knapsack Problem……..…… 39

3.5.1 Types of Knapsack Problems…….…… 40

3.5.2 Methods for solving Knapsack problems……..… 42

3.5.3 Dynamic Programming……... 42

3.5.4 The Branch and Bound Method……..…….. 48

3.5.5 Rounding Fractional Solutions.……..… 49

3.6 An Illustrative Example……..…… 50

CHAPTER 4:DATA COLLECTION AND ANALYSIS

4.1 Data Collection…… 67

4.2 Problem Formulation…… 69

4.3 Cutting Stock Algorithm…… 73

4.4Computational Procedures and Results…… 74

4.5Discussion…… 79

4.6Features of the Application…… 80

CHAPTER 5:CONCLUSION AND RECOMMENDATION

5.1 CONCLUSION…… 82

5.2 RECOMMENDATION…… 82

REFERENCES…… 83

APPENDIX A: OUTPUT OF SAMPLE DATA BY THE APPLICATION…….. 87

APPENDIX B - VISUAL BASIC CODES FOR THE CUTTING STOCK ……. 123

LIST OF TABLES

Table *Page*

Table 3.1: Simplex Tableau……..... 34

Table 3.2 Sample of possible patterns that can be generated…… 37

Table 3.3: Knapsack problems and their sub-problems …..…….…… 47

Table 3.4 Knapsack Sub-problems solutions……..……..…… 48

Table 3.5Order demands with quantity…….……..……..…… 50

Table 3.6 Knapsack problems and their sub-problems…..……..…… 55

Table 3.7 Knapsack sub-problems and their solutions…..…… 56

Table 3.8 Knapsack problems and their sub-problems…..……..… 59

Table 3.9 Knapsack sub-problems and their solutions…..……..… 60

Table 3.10 Knapsack problems and their sub-problems…..……..… 63

Table 3.11Knapsack sub-problems and their solutions…..……..… 64

Table 4.1 Order data from the company……..…… 68

Table 4.2 length of Strip after the Ripping process……..…… 68

Table 4.3 Company Strip quantities and cost utilized to meet the
orders in table 4.1……..……..…… 69

Table 4.4 Quantities of cut-pieces generated by application for the
five order demands for August to December, 2010……..…… 74

Table 4.5 Quantity and cost of Strips used by application to generate
quantities of Table 4.4.…… 75

Table 4.6Optimal Pattern generated and their usage for the August orders…… 76

Table 4.7Optimal Pattern generated and their usage for the September orders…… 77

Table 4.8Optimal Pattern generated and their usage for the October orders…… 77

Table 4.9Optimal Pattern generated and their usage for the November orders…… 78

Table 4.10Optimal Pattern generated and their usage for the December orders… 78

Table 4.11 Summary of Total cost of strips used by Company

and the application for set of orders……..… 79

LIST OF FIGURES

Figure *Page*

Figure 4.1: User interface for the Cutting stock application…….. 80

Figure 4.2: Output from the application ……..…….. 81

DEDICATION

To my family

ACKNOWLEDGEMENT

I thank my thesis advisor, Mr. KwakuDarkwah for his invaluable suggestions and supervision throughout the study.

I thank the staff of Rogersco Sawmill Limited, especially Mary Gyasi, Yawkwei, for providing the data for the study.

I would also like to thank my friends and course mates, Abdul Rasheed and Emmanuel Ofori, for their invaluable support

Last but not the least, I would like to express my gratitude to my family,especially my aunt, Mrs. Elizabeth Osei, Robert Kwakye, Nana KwadwoDuah and Mrs. Opoku, for their support which enabled me to accomplish the study.

CHAPTER ONE

INTRODUCTION

Economic resources are scarce and havecosts associated. The scarcity and cost associated with these resourcesgenerally impose certain constraints in their utilization. The effective management of these constraints aimedat minimizing the overall cost of input resources and the maximization of corresponding profitis the subject of optimization.Practical optimization is the art and science of allocating scarce resources to the best possible effect (Amponsah, 2006). Optimization techniques, a branch of mathematical programming, has enjoyed enormous appeal after World War II, both in the academia and in practice. Subsequently, this interest inspired numerous researches that sought to identify, analyze and substantiate new techniques for improving industrial and business processes. Currently, Optimization techniques have become an indispensable tool for industrial applications including resource allocation, scheduling, decision-making, etc. Optimization techniques have various branches and one such branch is linear programming.

The term "programming" in linear programming does not assume programming as used in the field of computer science to denote software development. Instead, it focuses on mathematical modeling and the requirement of a finite number of iterations to solve the model.This thesisfocuseson a special type of linear programming called the Cutting Stock Problem.

1.8 The Cutting Stock Problem

The Cutting Stock Problem is the problem of filling an order at minimum cost for specified numbers of lengths of material to be cut from given stock lengths of given cost (Gilmore et

al., 1961). The cutting-stock problem is an integer programming problem. However, since integer programming problems are known to be non-deterministic polynomial-time (NP) hard, the Cutting Stock Problem is formulated as linear programming (LP) problem by relaxing the integer requirements. After the LP optimal has been found, a rounding-upprocedureis used to get the integer programming (IP) optimum.It arises from many applications in industry including paper, glass, shoe-leather cutting, furniture, machine-building, etc.

A typology of Cutting problems by Dyckhoff (1990),classifies them into one-, two- and three-dimensional problems.One-dimensional cutting occurs when cutting for example pipes, cables, and steel bars. Two-dimensional problems are encountered in furniture, clothing and glass production. Not many three-dimensional (3D) cutting applications are known.

1.9 Background to Rogersco Sawmill Limited

Rogersco Sawmill Limited is located on the outskirts of Yawkwei; a town located about 4 kilometers from Konongo on the Accra – Kumasi highway. Rogersco cuts wood into various smaller sizes for various customer needs. The wood coming into the Sawmill is of long and wide pieces called lumber. The lumber is cut into strip boards and then into smaller pieces to meet customer demands. The small pieces of wood cut out of the lumber are called "cut-pieces". The initial operation performed on the lumber transforms it into long rectangularbeam of wood. The next operation, called Ripping, is theprocess of cutting these rectangular beams of wood along its cross-sectional width to get long but narrow pieces of woods which arecalled strip boards (or strip). The strips have width of 1- and 2-inch(es).In the next stage, the obtained strips are cut along their cross-sectional length into

desiredsmaller lengths. This process is called strip-cutting. Rogersco Sawmill does not cut stock primarily to meet customer demands, but to supply to their depot in Accra and therefore do not consider any "excess" cut as overproduction. This means that so long as there is lumber it will be processed. The main cut-pieces are 2"x6", 2"x4", 2"x3" and 2"x2". However they sometimes receive orders for other pieces like 1"x12", 2"x5" or 1"x6", but these are comparatively rare. All these cuts are made with a machine called Ban Mill and the cuts are all guillotine cuts. A guillotine cut means that each cut must go from one side of a rectangle straight to the opposite.

1.10 Problem Statement

The primary aim of every business is to optimize cost (to maximize profit or minimize the cost of operation) while meeting demands. In order to satisfy the demand of its clients and also to supply their depot, Rogersco Sawmill Limited cuts timber logs into stripboards and then into cut-pieces.

This thesis seeks to address the problem of finding optimal cutting patterns for cutting these strip boards into cut-pieces so that the total cost of the strip boards used to satisfy orders for cut-piecesis minimized.

1.11 Objective(s) of the Study

The objectives of this thesis are:

a) to model the Cutting Stock Problem (CSP) as a multi objective problem of Linear Programming and a Knapsack Problem.

b) to develop a computer application based on the Simplex algorithm and dynamic programming to solve the cutting stock problem.

c) to find optimal Cutting patterns using Simplex algorithm and dynamic programming techniques.

d) to minimize the total cost of stocks that are cut to meet order demands by customers.

1.12 Methodology

This thesis seeks to model the cutting stock problem as a multi objective optimization problem consisting of alinear programming master problem and a knapsack sub-problem. Data on the cross-sectional dimension of various sizes of strip boards, quantities of each strip board type and the total cost of the strip boards utilized to meet demand for smaller sizes, cross-sectional dimension of smaller sizes requested and their respective quantities per order and the sample orders for August 2010 to December 2010 will be obtained from Rogersco Sawmill Limited.

Methods to be employed to solve the Cutting Stock problem are the Simplex algorithm and dynamic programming techniques.

A computer application, based on these methods, is developed to solve the cutting stock problem. The application is developed using Visual Basic.Net 2005 edition. The KNUST library and the internet will provide other sources of information to this thesis.

1.13 Justification

The timber industry depends on the forest for its raw materials. However, widespread concerns, both locally and internationally, about the fast degradation of the forest and the

consequent ecological threat that deforestation poses to our environment and our survival, both presently and in the future, have limited access to these raw materials. Government agencies responsible for the protection and management of such forest as well as non-governmental bodies have adopted policies which seek to protect the forest and to control or manage access to its resources. These policies have contributed immensely to scarcity in the quantities of timber logs. Scarcity has made the cost of logs dearer and consequently high cost of orders for wood products from saw mills and therefore efficiency in the utilization of the timber logs in these saw mills, to meet various smaller sized demands, becomes essential. Cost is associated with these timber logs and the overall cost of customer orders will depend on the number and cost of timber logs that are cut to meet those orders. Efficient utilization means minimizing, as much as feasible, the quantities of logs that are used to satisfy customer demands. To this end, any approach that seeks the efficient utilization of these scarce timber logs cannot be over emphasized.

This thesis seeks to minimize the cost of strips needed to meet demand for cut-pieces of wood. This will help Rogersco Sawmill Limited to reduce the total cost of lumber that is processed to satisfy customer demands and its operational cost. Consequently, this will increase profit of the company since cost will be reduced. This could also help reduce the quantity of lumber that is cut from the forest and thereby help forest conservation efforts by government and other agencies.

1.14 Thesis Organization

Chapter One is the introduction which comprises the background to the problem, statement of the problem, objectives of the study, justification of the research, methodology and limitation.

Chapter Two deals with the review of literature on the cutting stock problem.

Chapter Three describes the methodologies used in this thesis. It comprises of the introduction, describing Linear Programming and basic terminologies associated with linear programming, showing how the cutting stock problem is formulated as a linear program, the concept of column generation, and the application of knapsack to solve such problems. The data used in the study and data analysis and results are considered in Chapter Four.

The summary and conclusions including a discussion of the policy implications of the study are presented in Chapter Five.

CHAPTER 2

LITERATURE REVIEW

2.1 A Review of the Cutting Stock Problem.

Industrial applications of cutting-stock problems for high production volumes arise especially when basic material is produced in large rolls or sizes that are further cut into smaller units. Within such disciplines as Management Science, Information and Computer Science, Engineering, Mathematics and Operations Research, problems of cutting of concrete and abstract objects appear under various specifications (cutting problems, knapsack problems, container and vehicle loading problems, pallet loading, bin packing, assembly line balancing, capital budgeting, etc.). In cutting problems, a large object must be divided into smaller pieces; in packing problems, small items must be combined to large objects. Most of these problems are considered NP-hard. This is done e.g. in paper and plastic film industries but also in production of flat metals like steel or brass. There are many variants and additional constraints arising from special production constraints due to machinery and process limits, customer requirements and quality issues.

The cutting stock problem was first formulated by Kantorovich (1939). Kantorovich was charged with the reorganization of the timber industry in the U.S.S.R., and as a part of his task he formulated a restricted class of linear programs and a method for their solution (Matousekand Gartner, 2007). In 1951 before computers became widely available, L.V. Kantorovich and V.A. Zalgaller suggested solving the problem of the economical use of material at the cutting stage with the help of linear programming. The proposed technique was later called the Column Generation method.

2.2 Historical background of the Cutting Stock Problem

Eisemann (1957) proposed an economical allocation of raws to machines and of setting up cuts in such a way as to produce the ordered quantities of final widths as the minimum overall trimming loss consistent with certain imposed restrictions, paying particular attention to rolls of materials for example paper, textiles, cellophane, metallic foil. In his formulation, "raws" referred to uncut rolls as inserted in the cutting machines, "finals" as cut rolls of widths specified in the orders and "cuts" as one simultaneous cutting by all preset knife edges of one "raw" roll into one or several "finals". Eisemann described the formulation and solution of two alternative variants of the trim problem simultaneously. One was a 1-dimensional cutting problem in which the interest was with the number of rolls of ordered finals that was cut. The material to be cut, in this case, was not unwound, but the cutting knives slice through the completely wound rolls of material. The second formulation was when a roll is unwound and, during unwinding, is sliced lengthwise by knives for a certain total length. This second formulation, according to Eisemann, was particularly useful in the case of expensive materials.

The most common approach for solving the Cutting Stock Problem is the linear programming approach which was first proposed by Gilmore and Gomory (1961). The objective that they considered for the Cutting Stock Problem was minimizing the cost (or minimizing the total number of the required stock, assuming that there is unique price for each object).

In the LP approach, the LP relaxation (the model obtained by relaxing the integrality constraints on variables) of the problem is considered, and solved instead; then a rounding procedure is used to get an integer solution. The difficulty of the LP relaxation approach is that there is a large number of cutting patterns, which can hardly be enumerated. The column generation method

proposed by Gilmore and Gomory was developed to overcome this difficulty. In this method, the cutting patterns are generated, during the process of solving the problem, through an auxiliary problem. The method starts with a set of simple patterns (to form the initial basis), then the solution is improved by removing a cutting pattern and generating a new one (same as pivoting procedure in simplex; the cutting pattern which is removed is the leaving variable and the new cutting pattern is the entering variable). The new cutting pattern is generated using the auxiliary problem which is easy to solve; knapsack problem normally serves as the auxiliary problem, and there are several methods for solving this problem. The new column is generated in a manner that it results in the most possible improvement in the solution (based on the same concept as choosing the entering variable to be the nonbasic variable with the most negative reduced cost in the simplex method).

After Gilmore and Gomory proposed the column generation method, many researchers used it for the Cutting Stock Problem.Follow-up research by Pierce (1964) focused on the use of this algorithm in the paper industry to solve the roll trim problem. Hahn (1968) considered the problem that arises when stock sheets contain flaws, and gave a dynamic programming algorithm. Sarker (1988) used this approach for solving one-dimensional Cutting Stock Problem. It is also noticeable that possible defects are also considered in this paper. In this paper having defects in items are acceptable, but defected items have less value.

In a latter paper, Dyson and Gregory (1974), both of whom were involved in the manufacturing of flat glass for use in the motor industry, the production of mirror and windows, stated that Gilmore et al's use of an auxiliary knapsack problem to solving the cutting stock problem is a simplification of the real problem and therefore is inadequate. This is because it only satisfied a wastage criteria without selecting the sequence in which the cutting patterns are to be processed,

a process they called pattern allocation. They proposed a two-stage approach. The first stage is where the cutting patterns are produced based on the Gilmore and Gomory method. The second stage involved the sequencing of the set of cutting patterns (pattern allocation) from the first stage, so that the number of discontinuities is minimized. They had two approaches to the pattern allocation stage, namely the two-stage approach and the heuristic approach. However, at the time of writing, the heuristic approach had not been implemented. The sequencing problem turns out to be of the travelling salesman type.

Christofides and Whitlock (1977) designed for their *n*-stage solution approach of the constrained CSP an enumerative procedure to generate the cutting patterns (columns) without any duplication due to symmetry or cut ordering.

Ferreira et al. (1990) also investigated a two-stage problem, which they called a two-phased problem. The authors adapted Haessler's sequential heuristic procedure, initially developed for a classic CSP, to a two-stage cutting process. At every step of the sequential procedure they tried to find a set of good intermediate rolls insuring a good pattern for the first stage and good patterns for the second.

Goulimis (1990) approach to the one-dimensional CSP starts with the generation of all feasible cutting patterns, usually making provision for such constraints as the minimum size of the trim, the number of cuts in a pattern and the number of different lengths in a pattern.

Lirov (1992) mentioned enumerative approach as another approach for solving the Cutting Stock Problem in his survey. Enumerative approach contains discrete optimization methods such as branch and bound or dynamic programming or a combination of these two.

As we previously discussed, the column generation method (LP approach) starts with a set of possible cutting patterns and tries to improve the obtained solution by replacing the current

patterns with better (more efficient) ones if there is any. In contrast the enumerative methods try to generate the best cutting patterns instead of improving them.

The most common method in the enumerative approach is solving a discrete optimization problem by branch and bound method. This method is normally done with splitting the feasible region into smaller sets; computing the (lower or upper) bounds; and eliminating the sets which cannot make any improvement in the solution (having a worse bound than the current solution). Obviously the procedure stops when all the remaining subsets have been shown to contain no better option.

Dynamic Programming is the most common enumerative method used for solving Cutting Stock Problem. In this method the objective of the problem is normally considered to be maximizing the total obtained value. A value is assigned to each item; each incoming object is cut in a manner that the total obtained value is maximized.

Maculan et al. (1992) proposed a column generation method to solve linear programming with bounding variable constraints, extending their results to the solution of integer problems.

Besides all the exact methods reviewed so far, there are a large number of inexact (heuristic) methods for the Cutting Stock Problem. Seth et al. (1986) developed a heuristic for one-dimensional Cutting Stock Problem. Vahrenkamp (1996) proposed an interesting heuristic based on the packing concepts for the Cutting Stock Problem. Chen et al. (1996) presented a simulated annealing procedure for the Cutting Stock Problem.

Winston (1994) used column generation approach to solve the CSP for Woodco and the minimum waste incurred was only 15 feet. The knapsack sub problem was solved using branch-

and-bound procedure; the master problem was solved by an advanced method of simplex method called the product form of the inverse.

Carvalho and Rodrigues (1995) follow a LP approach. Their problem, however, is subject to a technological restriction- Finished rolls of one type should comprise every intermediate roll. The restriction allows predefining a list of possible intermediate rolls. The authors reformulate an initial LP problem posed in terms of finished rolls into a LP problem in terms of intermediate rolls. A column generation technique with a regular knapsack as an auxiliary problem is applied.

Morabito and Garcia (1997) reported the problem of cutting rectangular plates into smaller ones in Brazilian hardboard industry. The problem was to determine the best patterns to be cut by an automated machine composed of a set of circular saws, device to move and hold the plates and loading and unloading stations. A particular two-phase column generation procedure was described for the cutting stock formulation of the hardboard industry. Each phase of the procedure was modeled as an integer program and solved by two alternative methods. The first was a dynamic programming based integer program and the second was a simple extension of the algorithm presented in Gilmore and Gomory (1963).

Hopper and Turton (1999) studied the problem consisting of packing rectangular items onto a rectangular object while minimizing the used object space. The packing process has to ensure that there is no overlap between the items, which are allowed to rotate by 90°.

Authors applied two genetic algorithms (GAs) to solve this problem. Both GAs were hybridized with a heuristic placement algorithm, one of which is the well known Bottom-Left routine. A second placement method has been developed which overcomes some of the disadvantages of the Bottom-Left rule. The two hybrid genetic algorithms were compared with heuristic placement algorithms.

Morabito and Arenales (2000) analyzed practical aspects of the application of a cutting stock model to a Brazilian company that manufactures furniture on a large scale with a high degree of standardization. The model was based on the classical approach of Gilmore and Gomory (1965) which combines a linear programming and a column generation procedure. Besides the two-stage and three-stage guillotine cutting patterns, authors also considered one-group guillotine patterns that improve the productivity of the cutting equipment. Examples derived from the furniture company was used to illustrate some of the trade-offs involved, in particular the trade-off between cutting simpler patterns and patterns that yield less waste material, but reduce the productivity of the cutting machine. Gradisar (2002), made an evaluation between the one-dimensional cutting stock problem (1D-CSP) algorithms which was the main reference for the authors of this paper.

Kalvelagen (2002) in his paper describes an implementation of the column generation algorithm using General Algebraic Modeling System(GAMS). The well-known cutting stock problem was used. The algorithm consists of 2 different models, a master problem and sub-problem which exchange information. A mixed integer problem for this problem was trivially formulated in GAMSonce they have enumerated all possible cutting patterns.

Puchinger et al. (2004) described a combined genetic algorithm/branch & bound approach for solving a real world glass-cutting problem. The GA (Genetic Algorithm) uses an order-based representation, which is decoded using a greedy heuristic. The B&B (Branch & Bound) algorithm was applied with a certain probability enhancing the decoding phase by generating locally optimal sub-patterns. Reported results indicate that the approach of occasionally solving sub patterns to optimality may increase the overall solution quality.

Johnston and Sadinlija (2004) created a new model which resolves the non-linearity between pattern variables and pattern run-lengths in the one dimensional cutting stock problem by a novel use of 0-1 variables. Belov et al. (2005) investigated robust branch-and-cut-and-price (BCP) algorithms, their theoretical properties and presented numerical results for BCP.

Reinaldo and Luciano (2007) described approaches to generate cutting patterns that minimize the cost or waste of material, considering different particular constraints associated with longitudinal (horizontal) and transversal (vertical) saws, head cuts (head cuts are the vertical guillotine cuts that divide the plate into two parts), book rotation (a complete turn of 180^{o}) and item unloading stations of the cutting machine. The method was based on dynamic programming recursive formulas combined with greedy constructive heuristics and the primal simplex algorithm.

Arbib and Marinelli (2007) reported the assortment and trim loss minimization problem arising in an Italian plant, operated by Pilkington, which produces glass parts for the automotive market. Glass cutting was organized in two phases: in phase I large rectangular sheets of the same type were obtained from a ribbon of flat glass and sent to warehouse and in Phase II sheets of various types were taken from warehouse and cut into smaller rectangular parts of various sizes in order to satisfy a given demand. In both phases, a trim loss occurs. In this study authors used heuristic algorithm based on a p-median model with additional constraints that take into account all the relevant shop floor requirements for solving the problem.

Fekete and Schepers (2001), gave comprehensive overview of Specialized algorithms for the two-dimensional bin-packing problem which presented several lower bounds on the solution value using, respectively, partitioning of rectangles in various classes and dual feasible functions.

Boschetti and Mingozzi (2003) presented a new lower bound that dominates the bounds of

Martello and Vigo (1998) and Fekete and Schepers (2001).Boschetti and Mingozzi (2003) generalized these bounds to the case where rectangles may be rotated 90 degrees. Considering the same variant of the problem, Dell'Amico et al. (2002) presented a lower bound and an exact branch-and-bound algorithm. Padberg (2000) presented an extended formulation and subjected it to polyhedral analysis, reaching a tighter LP relaxation.

Hadjiconstantinou and Christofides (1995) studied two-dimensional knapsack problem but were able to solve instances of only moderate size. Caprara and Monaci (2004) presented an approximation algorithm for the two-dimensional knapsack problem and developed four exact algorithms based on various enumeration schemes.

Yaodong and Yiping (2009) discussed a rectangular two-dimensional cutting stock problem in the steel bridge construction. It was the problem of cutting a set of rectangular items from plates with arbitrary sizes that lie in the supplier specified ranges, such that the necessary plate area was minimized. This paper presents a heuristic algorithm for two dimensional cutting stock problems in bridge construction. The heuristic algorithm used both recursive and dynamic programming techniques to generate patterns.

The one-dimensional Cutting Stock Problem becomes more difficult when there are a large number of objects of different sizes available, or the number of items (cut-pieces) is large. In such problem we have the difficulty of various sized objects, since strips may have different lengths. A similar problem is addressed by Belov and Scheithauer (2002), and a method based on combination of some enumerative methods is proposed.

CHAPTER 3

METHODOLOGY

3.1 General Overview of Cutting Stock Problem

The aim of Cutting Stock Problem is to minimize the total cost of stock length of given cost that is cut to fill an order for specified quantities of smaller lengths. This is achieved by generating optimal cutting patterns for the cutting of the stock lengths. Stock lengths have cost associated with them. The greater the quantity of stock lengths used in filling an order, the greater the cost of the order to the customer. This, essentially, means reducing the quantity of stock lengths used. It arises from many applications in industry.

The Cutting Stock Problem is essentially an integer programming (IP) problem. However, integer programming problems are known to be NP-hard and therefore, the Cutting Stock Problem is formulated as a linear program by relaxing the integer constraint. This makes Cutting Stock Problems amenable to linear programming methods of solution. The resultant LP optimum is rounded to get the IP optimum.The columns of the basis matrix represent all the cutting patterns that can be produced from the available stock length. The number of cutting patterns can be very large and this makes explicit enumeration of all feasible cutting patterns impractical. Therefore an initial set of feasible patterns is generated and used as the basis for the simplex method to solve for the dual variables.

An auxiliary problem arises in the simplex iteration where we choose the next column to enter basis.A column generation technique is applied to generate an entering column in the next simplex iteration. The column generation technique is formulated as a knapsack problem and solved using dynamic programming. Therefore, the Cutting Stock Problem is a multi-objective

problem comprising a master problem formulated as a linear programming problem and anauxiliary problem formulated as a knapsack problem.

3.2 Classification of Cutting stock problem

The Cutting Stock problems can be classified by the dimensions of the cutting object. This can beone-, two- or three-dimensional problems.

3.2.1 One dimensional cutting stock problem

The one-dimensional cutting stock problem is to obtain a given set of ordered lengths (patterns) from stock lengths. The objective is typically to minimize the total cost of stock materials used (material input). A cutting pattern describes how many items of each type are cut from a stock material.The one-dimensional cutting stock problem is defined by the following data. Let

$m =$ the number of smaller piece types

$L_k =$ the length of each stock material k

$l_i = length\ of\ ith\ piece, i = (1, \dots, m)$

$b_i = quantity of each piece length l_i ordered.$

$n = number of all cutting patterns$

$j = pattern j$

$x_j = number of times pattern j is applied in the solution$

$a_{ij} = the number of times piece i appears in pattern j$

$c_j = cost of stock length for pattern j$

the one dimensional model is as follows:

$$Z = Min \sum_{j=1}^{n} c_j x_j$$

$$s.t \sum_{j}^{n} a_{ij} x_j \geq b_i, \qquad i = 1, \ldots, m$$

$$x_j \in \mathbb{Z}_+, \qquad j = 1, \ldots, n$$

where x_j is the number of times pattern j is used, c_j is the cost of stock material used for cutting pattern j, a_{ij} is the number of l_i in pattern j and b_i is the quantity of l_i ordered.

To be a valid cutting pattern, a pattern must satisfy

$$\sum_{i=1}^{m} l_i a_{ij} \leq L_k$$

where L_k is the length of the *kth* stock material used to cut the pattern.

The huge number of patterns is not available explicitly for practical problems. Usually, necessary patterns are generated during a solution process, hence the term column generation. However, the number of different patterns in a solution cannot be greater than the number of stock lengths and is usually comparable with the number of piece types.

3.2.2 Two dimensional cutting stock problem

Another important variant of the cutting stock problem is the two-dimensional cutting stock problem. This variant can be divided into regular (rectangular, circular) and irregular shapes (Farley, 1988). Rectangular shapes can be obtained through guillotine or non-guillotine, oriented

or non-oriented cutting. An oriented cutting means that the lengths of rectangles are aligned parallel to length of the stock sheet.

A two-dimensional cutting stock problem can be defined as follows:

A set of rectangular stock sheets of $p$different types is available. For each type of sheet $S_p$we know its length $L_p$and widthW_p. From these sheets we have to cut smaller rectangular pieces of length $l_i$and widthw_i, $i = 1,\ldots,m$in order to satisfy a given demand for $d_i$pieces of each type. The objective is to minimize the total area of stock sheets required. A sequence of cuts of a sheet into rectangular pieces is a cutting pattern.

To formulate the two-dimensional cutting stock problem, we use the following notations:

$m =$ the number of smaller piece types

$\mathrm{Q} =$ set of all feasible two $-$ dimensional cutting patterns for all sheets $\mathrm{S_p}$

$d_i = quantity of each piece i ordered.$

$n = number of all cutting patterns$

$q = pattern q$

$x_q = number of times pattern q is used in the solution$

$a_{iq} = the number of times piece i appears in pattern q$

$c_q = cost of pattern q. It is the cost of sheet from which the pattern q was cut$

the two dimensional model is as follows:

$$Min \sum_{q=1}^{n} c_q x_q$$

$$s.t \sum_{j}^{n} a_{iq} x_q \geq d_i, \qquad i = 1, \ldots, m$$

$$x_q \geq 0 \text{and integer}, \forall q \in Q$$

Since, as in the one dimensional case, the set of patterns$Q$cannot be completely described except for very small problems, we develop a column generation scheme that can be summarized as follows:

Step 1. Generate an initial set $\bar{Q}$ of m cutting patterns, where each pattern contains one type of piece.

Step 2. Solve the linear relaxation of the above formulated problem considering only the variables corresponding to patterns in$\bar{Q}$.

Step3.for each type of sheetS_p, find non-negative integers $a_i, (i = 1, \ldots, m)$, by solving the knapsack sub-problem using dynamic programming:

$$z_p = Max \sum_{i=1}^{m} \pi_i a_i$$

$$s.t\{a_1, \ldots, a_m\} \text{is a feasible cutting pattern for } S_p,$$

where π is the vector of dual prices of the LP solution. If for some p, $z_p > C_p$, then the column corresponding to that solution is added to $\bar{Q}$and we return to Step 2 in order to solve the enlarged LP problem. Otherwise, the current solution is rounded to get an

integer solution and the process terminates. The question now is how to solve efficiently the sub-problem of Step 3.

3.2.3 Three dimensional cutting stock problem

Not many three dimensional applications are known.However, the closely related 3D bin-packing problem has many industrial applications, such as packing objects into shipping containers.

3.3 Strip Board Cutting as a One Dimensional Cutting Stock Problem.

The Strip boards are long narrow wood with rectangular cross-section.The width is usually 1 or 2 inches wide but may have variable lengths. The 1-inch strip boards are cut into one small type of length 12 inches. However, the 2-inch strips are cut into smaller sizes with various lengths of 2, 3, 4 and 6 inches. We refer to these smaller pieces as cut-pieces.Given an order for quantities of cut-pieces we want to find optimal ways to cut the 2- inch wide strip boards along their length to satisfy these demands at minimum cost of the total strip boards utilized. The cutting of the strip boards therefore can be described as a one-dimensional cutting stock problem.

3.3.1 General Formulation of the Strip Board Cutting

In this section we propose a Linear Programming (LP) model for optimal cross-sectional cutting of strip boards of wood into smaller cut-pieces to satisfy demand for these cut-pieces. As mentioned in chapter One, the strip-cutting problem for a given strip board of wood is the problem of determining an optimal cross-sectional cutting pattern for each strip of wood such that the corresponding number of cut-pieces of each type obtained is at least equal to the

corresponding demand for that cut-piece type and the expected total cost of strip board utilized is minimized.

Let

m:number of cut-piece type. This is also the number of constraints.

n:number of patterns.

k:number of different strip board lengths.

l_i: length of $i$thcut-piece , $i = 1, \dots, m$.

L_j: length of strip board $j, j = 1, \dots, k$.

j:the$j$th pattern or variable.

x_j:number of times pattern j is used in the solution. These are the decision variables for the LP

a_{ij}:number of cut-piece i in pattern j. This is the coefficients in the demand constraints.

b_i:the demand for cut-piece i

c_j:cost of the strip board used by pattern j

P: $(p_1, \dots, p_m)^T$,a cutting pattern

y: $(y_1, \dots, y_m)$, dual variables of cut-pieces

Z : total cost of stocks length used

Generally, the linear program is

$$Min\ Z = \sum_{j=1}^{n} c_j x_j \qquad (3.1)$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j \ \geq b_i \quad \forall\, i = 1, \dots, m \qquad (3.2)$$

$$x_j \geq 0 \qquad (3.3)$$

The objective function (3.1) is the total cost of stock lengths used. Constraint (3.2) requires that the quantity produced is at least equal to quantity demanded for each item i. Constraint (3.3) is the non-negativity restriction on the decision variables.

The dual of this formulation is given as

$$Max \sum_{i=1}^{m} y_i b_i \qquad (3.5)$$

$$s.t \sum_{i=1}^{m} a_{ij} y_j \leq c_j, \qquad \forall j = 1, \dots, n \quad (3.6)$$

$$y_i \geq 0 \quad (3.7)$$

A solution for the dual variables is given as

$$y = C_B B^{-1} \quad (3.8)$$

where B^{-1} is the inverse of the basis matrix and C_B is the cost vector corresponding to the columns of the basis matrix.

For a minimization problem, at each iteration in the Simplex algorithm for the solution of (3.5), (3.6) and (3.7), we find a feasible pattern $P = (p_1, \dots, p_m)^T$ that cuts from strip L_j with cost c_j such that

$$c_j - yP < 0.$$

However, for the Cutting Stock Problem, the list of patterns can be very large, and therefore we employ the Column generation technique that generates entering pattern by formulating this auxiliary problem as a knapsack problem.

3.3.2 Formulation of the auxiliary Knapsack Problem.

Let $P = (p_1, \dots, p_m)^T$ be a pattern that cuts from a strip L_j with cost c_j, andp_i be number of item i in the pattern. Then

$$c_j - yP < 0$$

that is

$$yP > c_j \,. \qquad (3.9)$$

$P_j$must also be a feasible pattern which means that

$$l_1 p_1 + l_2 p_2 + \cdots + l_m p_m \leq L_j (3.10)$$

$$p_i \geq 0 \; and \; strictly \; integer \; (3.11)$$

Inequality (3.10) requires that the total length of the items cut must not exceed the length of the strip from which the pattern is cut and (3.11) requires only integral cuts are made. The requirements (3.9), (3.10) and (3.11) must be satisfied for a pattern *P* to enter basis and they are what we need to generate the next column.

The column generation technique of (3.9), (3.10) and (3.11) is a single constrained linear programming problem. However, this single constraint has an additional requirement as strictly integer which cannot be relaxed, and therefore the auxiliary problem becomes a Knapsack problem. The Knapsack problem for the cutting stock problem is formulated as follows:

Let y_i be the dual variables from (3.8) and p_i be non-negative integers, l_i be the length of item i, $(i = 1, \dots, m)$, L_j be strip from which a pattern *j*is cut and c_i be cost of thestripL_j. We use (3.9), (3.10) and (3.11) to define the problem as

$$z = Max \sum_{i=1}^{m} y_i p_i \, (3.12)$$

$$s.t \sum_{i=1}^{m} l_i p_i \leq L_j \qquad (3.13)$$

$$p_i \geq 0 \; and \; integer \quad (3.14)$$

The solution is a set of non-negative integers $(p_1, \ldots, p_m)$ that maximizes (3.12) subject to (3.13) and (3.14). If

$$z > c_i$$

then the pattern P with components $(p_1, \ldots, p_m)^T$ enters basis. If such a column cannot be found then the iteration terminates and current solution is optimal. Otherwise, we proceed and use this new pattern P as the entering variable and compute the leaving variable of the linear programming problem with the minimum ratio test

$$\theta = \min \left(\frac{RHS}{coefficient\ of\ new\ pattern\ P}\right)$$

This is a component-wise division between the current solution or right-hand-side and the new pattern. This minimum determines the leaving variable after which the simplex proceeds normally. The simplex iteration continues until no pattern is found that can improve the current solution. In such case, the current solution is our optimum.

Formulation of an Initial Feasible Basis

The number of constraints corresponds to the number of cut-piece typeto cut. Therefore, since the number of cut-piecetypes is m, number of constraint is also m. Each constraint represents the total quantity of each cut-piecetype that will be cut from the patterns considered in the solution.We create m initial patterns and define the constraints coefficients of the linear

programming problem as the components of these initial patterns. Since we define initial m patterns, $m = n$.

The m initial cutting patterns are created such that the ith pattern cuts only thecut-piecel_i $(i = 1, \cdots, m)$. Arrange the stock lengths such that $L_1 > L_2 > \cdots > L_k$. Choose stock length $L_j$for which $L_j > l_i$ and define the *jth*pattern to be the one cutting

$$a_{ii} = \left\lfloor \frac{L_j}{l_i} \right\rfloor$$

$$a_{ik} = 0, i \neq k$$

$$i = 1, \ldots, m \; and \; k = 1, \ldots, m.$$

where$\lfloor \quad \rfloor$ represents largest integer less than the value in the bracket. The cost of the *jth* pattern will be the cost of the stock length L_j from which the ith activity cuts the piece of lengthl_i. The m initial patterns are as follows:

Pattern 1 is $a_{11} = \lfloor L_j/l_1 \rfloor, a_{21} = 0, \cdots, a_{m1} = 0$

Pattern 2 is $a_{12} = 0, a_{22} = \lfloor L_j/l_2 \rfloor, \cdots, a_{m2} = 0$

$\vdots$

Pattern m is $a_{1n} = 0, a_{2n} = 0, \cdots, a_{mn} = \lfloor L_j/l_m \rfloor$

Our initial m patterns are

Pat1	Pat2		Pat m
a_{11}	0	$\cdots$	0
0	a_{22}	0	0
$\vdots$	$\vdots$	$\ddots$	$\vdots$
0	0	$\cdots$	a_{mn}

The Linear Programming problem then becomes

$$Min\ Z = \ c_1x_1 + c_2x_2 + \cdots + c_nx_n \qquad (3.15)$$

subject to

$$\begin{aligned} a_{11}x_1 + 0x_2 + \cdots + 0x_n &\geq b_1 \\ 0x_1 + a_{22}x_2 + \cdots + 0x_n &\geq b_2 \\ &\vdots \\ 0x_1 + 0x_2 + \cdots + a_{mn}x_n &\geq b_m \end{aligned} \qquad (3.16)$$

$$x_1, x_2, \cdots, x_n \geq 0 \qquad (3.17)$$

The dual of this formulation is given as

$$Max\ \ b_1y_2 + by_2 + \cdots + b_my_m \qquad (3.18)$$

subject to

$$\begin{aligned} a_{11}y_1 + 0y_2 + \cdots + 0y_m &\leq c_1 \\ 0y_1 + a_{22}y_2 + \cdots + 0y_m &\leq c_2 \\ &\vdots \\ 0y_1 + 0y_2 + \cdots + a_{nm}y_m &\leq c_n \end{aligned} \qquad (3.19)$$

$$y_1, y_2, \cdots, y_m \geq 0 \qquad (3.20)$$

3.4 Methods of Solution For the Cutting Stock Problem

The Cutting Stock problem is formulated as a linear programming problem and it is solved using the revised Simplex method. The auxiliary knapsack problem that finds the next entering column is solved by dynamic programming or branch-and-bound algorithms. After the LP optimum is found, rounding-up proceduresare used to find the integer programming optimum.

3.4.1 Linear Programming

Linear Programming (LP) is a type of optimization technique used for economic allocation of 'scarce' resources to several competing activities on the basis of a given criterion of optimality. The phrase 'scarce resources' means resources that are limited in availability. The criterion of optimality generally is either performance, return on investment, profit, cost, utility, time, distance, etc. The term "linear" is used because all the relations among the variables are linear. On the other hand, the word "programming" refers to modeling and solving a problem mathematically that involves the economic allocation of limited resources by choosing a particular course of action or strategy among various alternative strategies to achieve the desired objective.

Assumptions of Linear Program

a) **Linearity (or Proportionality):** All relationships in the LP model must be linear

b) **Additivity:** The value of the objective function for the given values of decision variables must be equal to the sum of the contributions (profit or cost) earned from each decision variable and the total sum of resources used, must be equal to the sum of the resources used by each decision variable.

3.4.2 General Formulation of a Linear Program

Notations:

m:number of constraints.

n:number of variables.

x_j:decision variables

a_{ij}:coefficients of variable j in constraint i

b_i:right-hand-side coefficients for constraints i

c_j:objective function coefficients of the variable j

A: matrix (with m rows and n columns) of the coefficients of the variables in the constraints.

The LP model is

Maximization:

$$Max \sum_{j=1}^{n} c_j x_j$$

$$s.t \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \qquad \forall i = 1, \dots, m \quad (demand\ constraints)$$

$$x_j \geq 0$$

Minimization:

$$Min \sum_{j=1}^{n} c_j x_j$$

$$s.t \sum_{j=1}^{n} a_{ij} x_j \geq b_i, \qquad \forall i = 1, \dots, m \quad (demand\ constraints)$$

$$x_j \geq 0$$

Examples of Linear Programs

a) **The Diet Problem.** There are $m$different types of food, $F_1, \ldots, F_m$, that supply varying quantities of the $n$nutrients, $N_1, \ldots, N_n$, that are essential to good health. Let $c_j$be the minimum daily requirement of nutrient, N_j. Let b_i be the price per unit of food, F_i. Let a_{ij}be the amount of nutrient $N_j$contained in one unit of food F_i. The problem is to supply the required nutrients at minimum cost.

Let $y_i$be the number of units of food $F_i$to be purchased per day. The cost per day of such a diet is

$$Minimize\ Z = b_1 y_1 + \cdots + b_m y_m$$

The amount of nutrient $N_j$contained in this diet is

$$a_{1j} y_1 + \cdots + a_{mj} y_m, \quad \forall\, j = 1, \ldots, n$$

We do not consider such a diet unless all the minimum daily requirements are met, that is, if and only if$a_{1j} y_1 + \cdots + a_{mj} y_m \geq c_j, \quad \forall\, j = 1, \ldots, n$

$$y_1 \geq 0, y_2 \geq 0, \ldots, y_n \geq 0$$

b) **The Transportation Problem:**There are I ports, or production plants, $P_1, \ldots, P_I$, that supply a certain commodity, and there are J markets, $M_1, \ldots, M_J$, to which this commodity must be shipped. Port P_i possesses an amount s_i of the commodity$(i = 1, \ldots, I)$, and market M_j must receive the amount r_j of the commodity $(j = 1, \ldots, J)$. Let bij be the cost of transporting one unit of the commodity from port P_i to market M_j. The problem is to meet the market requirements at minimum transportation cost. Let y_{ij} be the quantity of the commodity shipped from port P_i to marketM_j. The total transportation problem is

$$\text{Minimize } Z = \sum_{i=1}^{I} \sum_{j=1}^{J} y_{ij}\, b_{ij}$$

The amount sent from port P_i is $\sum_{j=1}^{J} y_{ij}$ and since the amount available at port $P_i$is s_i, we must have

$$\sum_{j=1}^{J} y_{ij} \leq s_i \qquad \forall\, i = 1, \ldots, I$$

The amount sent from port M_j is $\sum_{i=1}^{I} y_{ij}$ and since the amount required there is r_j, we must have

$$\sum_{j=1}^{I} y_{ij} \geq r_i \qquad \text{for } j = 1, \ldots, J$$

$$y_{ij} \geq 0 \;\forall\, i = 1, \ldots, I \text{ and } j = 1, \ldots, J$$

c) **The Investment Problem:** At our disposal is a sum S units of money which may be invested in various activities, each of them producing a certain benefit. Let us denote by a_j, $1 \leq j \leq n$, the sum invested in the *j-th* activity. The above problem can be modeled using linear programming, that is;

Maximize

$$\sum_{j=1}^{n} a_j X_j$$

Subject to

$$\sum_{j=1}^{n} X_j \leq S \text{ and } X_j \geq 0$$

3.4.3 The Simplex Method

The simplex method is the most common way to solve large Linear Programming problems. Simplex is a mathematical term. The underlying concepts are geometrical, but the solution algorithm, developed by George Dantzig in 1947, is an algebraic procedure.The simplex method finds the most attractive corner of the feasible region to solve the LP problem. Any LP problem having a solution must have a optimal solution that corresponds to a corner, although there may be multiple or alternative optimal solutions.Simplex usually starts at the corner that represents doing nothing. It moves to the neighboring corner that best improves the solution. It does this over and over again, making the greatest possible improvement each time. When no more improvements can be made, the most attractive corner corresponding to the optimal solution has been found.

General Simplex Formulation

In general, the simplex algorithm is a method for solving linear programs in the following form,

$$\text{Maximize} c_1 x_1 + \cdots + c_n x_n$$

subject to the constraints

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \leq b_1$$

$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \leq b_2$$

$$\vdots$$

$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \leq b_m$$

$$x_1, x_2, \cdots, x_n \geq 0$$

The constraints

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \leq b_i$$

can be written as

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n + x_{n+i} = b_i$$

where $x_{n+i} \geq 0$ is a slack variable.

The new variables x_{n+i}, would be assigned zero cost coefficients in the objective function, i.e. $c_{n+i} = 0$.

In matrix notations, the standard form of a linear programming problem be represented by an $m \times n$ matrix

$$\mathbf{A} = \begin{pmatrix} a_{11}a_{12} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1}a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

together with an *n*-vector of "costs" $\boldsymbol{C} = (c_1, \cdots, c_n)$ and an *m*-vector of "right-hand sides" $\boldsymbol{b} = (b_1, \cdots, b_m)$. The variables can also be grouped into an *n*-vector:

$$\boldsymbol{X} = (x_1, \cdots, x_n, x_{n+i})$$

Then the entire linear program can be written as follows:

$$Maximize \boldsymbol{C}^T \boldsymbol{X}$$

$$subject\ to\ \mathbf{AX} = \mathbf{b}$$

$$\mathbf{X} \geq \mathbf{0}$$

Steps of the Simplex Method

STEP 1: Formulate the LP and construct a simplex tableau.

Add slack variables to represent unused resources, thus eliminating inequality constraints. Construct the simplex tableau, i.e. a table that allows you to evaluate various combinations of resources to determine which mix of resources will most improve your solution. Use slack variables in the starting basic variable mix. Table 3.1 shows a general construction of the simplex tableau. Row 2 represents the variables including slack variables. Variable $x_{n+1}, \cdots, x_{n+m}$ are the slack variables. Row 1 is the cost coefficient of each variable in the objective function. Row 3 to row m+2 are the constraint coefficients. Column 1 represents coefficients of basic variables. Column 2 shows the basic variables. Column labeled "RHS" shows the values on the right-hand-side of the constraints and is also the solution column. Column labeled "θ" shows the exit ratios.

Table 3.1 Simplex tableau

	c_j	c_1	...	c_n	0	...	0		
Basic variable coefficients	Basic variables	x_1	...	x_n	x_{n+1}	...	x_{n+m}	RHS	θ
0	x_{n+1}	a_{11}	...	a_{1n}	1	...	0	b_1	
⋮	⋮	⋮	⋮	⋮	⋮	...	⋮	⋮	
0	x_{n+m}	a_{m1}	...	a_{mn}	0	...	1	b_m	
	z_j	z_1	...	z_n	z_{n+1}	...	z_{n+m}		
	$c_j - z_j$								

STEP 2: Find an Initial Feasible Solution

Find an initial basic feasible solution (bfs). If none is found, then the model is infeasible, so exit.

STEP 3:Find the Entering Variable

Find the entering variable and mark the top of its column with an arrow pointing down. This is the pivot column. The entering variable is the current non-basic variable that has the largest positive $c_j - z_j$coefficient. This is based on the largest coefficient rule. If no such coefficient exists, this indicates that one or more constraints are unbounded.

STEP 4: Find the Leaving Variable

Applythe minimum ratio test (θ) to determine the leaving basic variable. This test determines which constraint most limits the increase in the value of the entering basic variable. The most limiting constraint is the one whose basic variable is driven to zero first as the entering basic variable increases in value.

$$\theta = \min\left(\frac{RHS}{coefficient\ of\ pivot\ column}\right)$$

If the coefficient of the entering basic variable is ≤ 0 we do not evaluate those. The leaving basic variable is associated with the row that has the minimum value of the ratio test. This row is called the pivot row. The tableau element where the pivot column and pivot row intersect is the pivot element. In the column entitled basic variable, replace the leaving variable listed for the pivot row by the entering basic variable. If the pivot element is not +1, then we divide all of the elements in the pivot row by the pivot element to obtain a +1 in the pivot element position.

STEP 5: Perform Gaussian operation on the tableau

Perform Gaussian elimination on the table to eliminate all the coefficients in the pivot column except the pivot element. This will bring the entering variable into basis. To remove the pivot column element in some row k,

$$(new\ row\ k) = (row\ k) - (pivot\ column\ coefficient\ in\ row\ k) \times (pivot\ row)$$

After the Gaussian operation, we have reached the second basic feasible solution. This process of moving from one basic feasible solution to the next is called pivoting.

STEP 6: Repeat STEP 2 to STEP 5 until no improvement can be found.

3.4.4 Column Generation

Column generation deals with adding variables to a master problem. It is one of the most used methods in real life with lots of applications including the cutting stock problems and bin-packing problems.

In solving linear programming problems using the Simplex algorithm, we enumerate all the decision variables and store them before setting our initial tableau. We then determine our initial basic feasible solution. We check the non-basic variables to find one with the most negative reduced cost to enter basis. However, a problem arises when we have extremely many variables.

For example, we have a demand to cut 511 pieces of 9-inch wood, 301 pieces of 8-inch wood, 263 pieces of 7-inch wood and 383 pieces of 6-inch wood from a given stock of 20-inch wood. Possible feasible cutting patterns for this instance are given in Table 3.2, assuming unlimited stock of 20-inch stock. Table 3.2 lists 10 of the possible feasible patterns that can be cut from each stock sheet:

Table 3.2 Sample of possible patterns that can be generated.

	Possible Patterns (Decision variables)									
Small length	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
9”	2	0	0	0	0	1	1	1	0	0
8”	0	2	0	0	0	1	0	0	1	1
7”	0	0	2	0	1	0	1	0	0	1
6”	0	0	1	3	2	0	0	1	2	0

This list is only a subset of the possible patterns that can be generated. This clearly indicates that many linear programs are too large to consider all the variables explicitly. Since most of the variables will be non-basic and assume a value of zero in the optimal solution, only a subset of variables need to be considered in theory when solving the problem. Column generation leverages this idea to generate only the variables which have the potential to improve the objective function, i.e., to find variables with negative reduced cost. The reduced cost is the reduction in objective function if non-basic variable is increased by one unit. Gilmore et al (1961) described this sub-problem as a Knapsack problem. The objective function of the sub-problem is the reduced cost of the new variable with respect to the current dual variables, and the constraints require that the variable obey the naturally occurring constraints. The master problem is solved to obtain dual prices for each of the constraints in the master problem. In a minimization problem, a pattern enters basis if

$$C_j - Z_j < 0$$

Therefore, we use this information to generate the entering pattern. Let $P_j = (a, b, c, d)^T$ be the pattern to enter basis and y be set of dual prices obtained from solving the initial master LP problem. $Z_j = yP_j$, soP_j enters basis if and only if

$$C_j - yP_j < 0$$

$$ie \;\; yP_j > C_j \;\; and \; P_j \; is \; a \; feasible \; pattern.$$

A feasible pattern means that

$$l_1 a + l_2 b + l_3 c + l_4 d \leq L_j (14)$$

$$a, b, c, d \geq 0 \; and \; strictly \; integer \;\; (15)$$

These requirements (14) and (15) must be satisfied for a pattern P_j to enter basis. The sub-problem resembles a single constrained linear programming problem. However, this single constraint has an additional requirement as strictly integer, in which case we cannot solve by linear programming method. The sub-problem is therefore treated as a Knapsack problem and solved using the Branch-and-Bound algorithm or Dynamic programming. This variable is then added to the master problem, and the master problem is re-solved. Re-solving the master problem will generate a new set of dual values, and the process is repeated until no negative reduced cost variables are identified. If sub-problem returns a solution with non-negative reduced cost; we can conclude that the solution to the master problem is optimal.

The column generation method is summarized below.

1. Initialize the procedure: Let B be a $m \times m$ basis matrix, with a collection of some (feasible) patterns.
2. Solve the following (auxiliary) problem:

$$C_B B^{-1} A$$

$$s.t \; l^T A \leq L$$

$$A \geq 0 \; and \; integer$$

whereC_B (vector of cost coefficient of the basic variable); and B^{-1} (inverse of the basic matrix) are known; A is a (column) vector which contains the decision variables of the problem. l is a vector equal to $(l_1, \ldots, l_m)^T$ and L is a constant. Let A^* denotes the optimal solution of problem.

3. If $C_B B^{-1} A \leq c$ then the current solution (cutting patterns) is optimal (because reduced costs, $c - C_B B^{-1} A^*$ are non negative for all non basic variables).
4. If $C_B B^{-1} A > c$ then A^* enters the basis (since we maximized $C_B B^{-1} A$, A^* is the most profitable cutting pattern, i.e. it has the most negative reduced cost).
5. Find the leaving variable by the ratio test.
6. Form the new basis, and go back to step 2.

3.5 The Knapsack Problem

The knapsack problem is particularly a simple integer program: it has only one constraint. Furthermore, the coefficients of this constraint and the objective are all non-negative. The following is an example of a knapsack problem:

$$Maximize\ 3x_1 + 3x_2 + 2x_3 + 2x_4$$

$$subject\ to\ 9x_1 + 8x_2 + 7x_3 + 6x_4 \leq 20$$

$$x_j \geq 0\ \&\ integer$$

The traditional story is that there is a knapsack (here of capacity 20). There are a number of items (here there are four items), each with a size and a value (here item 2 has size 8 and value 3). The objective is to maximize the total value of the items in the knapsack.

3.5.1 Types of Knapsack Problems

Knapsack problems are found in various situations. They come in various types including single and multiple-constrained knapsacks, multidimensional knapsacks, multiple choice knapsacks, single and multiple objective knapsacks, integer, linear, non-linear knapsacks, deterministic and stochastic knapsacks, knapsacks with convex or concave objective functions, etc.

The Single 0-1 Knapsack Problem

This is a 0-1 knapsack problem, pure integer programming with single constraint which forms a very important class of integer programming. The 0-1 Knapsack Problem (KP) can be mathematically formulated through the following integer linear programming. Given a set of n items, let

P_j = profit of item j

W_j = weight of item j

c = capacity of knapsack

$$Max \sum_{j=1}^{n} P_j X_j$$

$$s.t \sum_{j=1}^{n} (W_j X_j) \leq c$$

$$X_j \in \{0,1\}, \qquad j = 1, \dots, n$$

where

$$X_j \begin{cases} 1 & \text{if item } j \text{ is selected} \\ 0 & otherwise \end{cases}$$

Multiple Knapsack Problems

An important generalization of the 0-1 knapsack problem is the 0-1 Multiple knapsack problem arising when *m* containers, of given capacities $c_i, (i = 1, \dots, m)$ are available. By introducing binary variablesX_{ij}, taking value 1 if item j is selected for the container i and value 0 otherwise, we obtain the formulation

$$Max \sum_{i}^{m} \sum_{j=1}^{n} P_j X_{ij}$$

$$s.t \sum_{j=1}^{n} (W_j X_j) \leq c_i$$

$$\sum_{i=1}^{m} X_{ij} \leq 1$$

$$X_{ij} \in \{0,1\}, \qquad j = 1, \dots, n \ \ and \ i = 1, \dots, m$$

where

$$X_{ij} \begin{cases} 1 & \text{if item } j \text{ is assigned to knapsack i} \\ 0 & otherwise \end{cases}$$

The generalization arising when the item set is partitioned into subsets and the additional constraint is imposed that at most one item per subset is selected is called the Multiple-Choice Knapsack Problem. The multi choice knapsack problem is defined as in knapsack problem with additional disjoint multiple choice constraint. The general description of the problem is given as follows: There is one knapsack with limited capacity. Objects to be packed in the knapsack are classified into multiple mutually exclusive classes. Within each class, there are several different items. The problem is to select some items from each class so as to minimize the total cost while

the total size of the items does not exceed the limited capacity of the knapsack. This problem is of a generalized carryout problem and is NP-hard.

3.5.2 Methods for solving Knapsack problems

There are two basic methods for solving the 0-1 knapsack problems *(KP):* Theses are dynamic programming and Branch-and-Bound methods. However the use of meta-heuristics including genetic algorithm, tabu-search and simulated annealing have been used to solve large scaleproblems.

3.5.3 Dynamic Programming

Dynamic programming is a method for efficiently solving a broad range of search and optimization problems which exhibit the characteristics of overlapping sub-problems and optimal substructure.

Overlapping Sub-problems

A problem is said to have overlapping sub-problems if it can be broken down into sub-problems which are reused multiple times. This is closely related to recursion. Consider the Fibonacci numbers

$$Fib(n) = Fib(n-1) + Fib(n-2)$$

$$Fib(0) = 0, \qquad Fib(1) = 1$$

The problem of calculating the n^{th} Fibonacci number exhibits overlapping sub-problems because calculating Fib(n) depends on both Fib(n-1) and Fib(n-2). At the k^{th} stage we only need to know

the values of $Fib(k-1)$and $Fib(k-2)$, but we wind up calling each multiple times. With dynamic programming, we can calculate the numbers we need for the next step, removing the massive redundancy.

Memoization

In dynamic programming, we write out a recursive formula that expresses large problems in terms of smaller ones and then use it to fill out a table of solution values in a bottom-up manner, from smallest sub-problem to largest. The formula also suggests a recursive algorithm, solving the same sub-problems over and over again. We need a more intelligent recursive implementation, one that remembers its previous invocations and thereby avoids repeating them. On the knapsack problem, such an algorithm would use a hash-table to store the values of F(s) that had already been computed. At each recursive call requesting some F(w), the algorithm would first check if the answer was already in the table and then would proceed to its calculation only if it wasn't. This is called memoization. Memoizationis another way to deal with overlapping sub-problems in dynamic programming. After computing the solution to a sub-problem we store it in a hash table. Subsequent calls to the sub-problem just do a table lookup.If solution of the sub-problem is not found in the table, compute the solution and add it to the list of sub-problems.

Optimal Substructure

A problem is said to have optimal substructure if the globally optimal solution can be constructed from locally optimal solutions to sub-problems. Optimal solution to problem consists of optimal solutions to sub-problems.

Solving Knapsack Problems with Dynamic Programming

Consider the knapsack problem.

$$Max \sum_{i=1}^{m} c_i a_i$$

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

$$a_i \geq 0 \; and \; integer$$

Each item i has a weight w_i pounds and a value c_i. The goal is to place items in our knapsack so that we get the maximum value without exceeding the weight limit of W pounds. Let $s = 0,1, \ldots W$ be set of weights, then

$$F(s) = max\left\{\sum_{i=1}^{m} c_i a_i : \sum_{i=1}^{m} w_i a_i \leq S \, , a_i \geq 0 \; and \; integer\right\}$$

$F(s)$represents the maximum value that can be attained by putting s in the knapsack. We can calculate $F(s)$ for $s = 0,1, \ldots W$ using$F(s) = 0$ and

$$F(s) = max\{c_i + F(s - w_i)\}$$

$$F(s) = -\infty \; \forall \, s < 0$$

A knapsack of weight at most s is obtained by first filling the knapsack with weight at most $s - w_i$, and then adding an item of weight w_i . The knapsack of weight at most $s - w_i$is filled such that we obtain the maximum value, $F(s - w_i)$and the ith item chosen so that the total value $F(s - w_i)$+ c_i is maximized.$F(s - w_i)$becomes a subproblem.

Backtracking

Backtracking is a systematic way to go through all the possible configurations of a search space. In the general case, we assume our solution is a vector$a = (a_1, \dots, a_n)$where each element a_i is selected from a finite ordered set S_i. We build a partial solution of length k,$a = (a_1, \dots, a_k)$ and extend it by adding another element. After extending it, we test whether what we have so far is still possible as a partial solution.If it is still a candidate solution then we stop, else we delete a_k and try the next element from S_k.Recursion can be used for elegant and easy implementation of backtracking. It can easily be used to iterate through all subsets or permutations of a set. It ensures correctness by enumerating all possibilities.

Backtrack Algorithm:

Step 1: Define a partial solution of length k, $a = (a_1, \dots, a_k)$from S_i. If a is a solution stop else gotoStep 2

Step 2: Expand the solution by adding one more element i.e. k = k + 1.

Step 3: Compute a new S_i

Step 4: While S_i is not empty, test whether a is still a partial solution of S_i. If not we remove a_k and repeat with the next element in S_i

In a Knapsack computation, after a maximum, $F(s)$, has been found, we use backtracking to determine the item set that gave that maximum and their corresponding frequencies, $a_1, \dots, a_m$. In the Cutting Stock case this process helps to form the entering pattern for entry into the Simplex algorithm. The frequencies will be the components of the pattern.

Dynamic Programming Example:

$$Maximize\ 11x_1 + 7x_2 + 5x_3 + x_4$$

$$subject\ to\ 6x_1 + 4x_2 + 3x_3 + 1x_4 \leq 15$$

$$x_j \geq 0\ \&\ integer$$

$F(0) = 0, F(s < 0) = -\infty$

$$F(15) = max\{11 + F(15-6), 7 + F(15-4), 5 + F(15-3), 1 + F(15-1)\}$$

$$= max\{11 + F(9), 7 + F(11), 5 + F(12), 1 + F(14)\}$$

Therefore to find F(15) we have to first find sub-problems F(9), F(11), F(12) and F(14). These problems will also have sub-problems. Table 3.3contains problems and their corresponding sub-problems with items that add to the sub-problems.The first column contains the subset of the weight of the knapsack. The first row of Columns 2 to 4 contain items that are included in the knapsack to get maximum value, whiles the remaining rows contain the maximum value that can be attained by adding the corresponding item. The maximum value is given as the sum of the value of the item and the maximum value for the sub-problem to which the item is being added.

Table 3.3: Knapsack problems and their sub-problems.

Knapsack Sub-problems	Items weights 6	4	3	1
0	0	0	0	0
1	11 + F(-5)	7 + F(-3)	5 + F(-2)	1 + F(0)
2	11 + F(-4)	7 + F(-2)	5 + F(-1)	1 + F(1)
3	11 + F(-3)	7 + F(-1)	5 + F(0)	1 + F(2)
4	11 + F(-2)	7 + F(0)	5 + F(1)	1 + F(3)
5	11 + F(-1)	7 + F(1)	5 + F(2)	1 + F(4)
6	11 + F(0)	7 + F(2)	5 + F(3)	1 + F(5)
7	11 + F(1)	7 + F(3)	5 + F(4)	1 + F(6)
8	11 + F(2)	7 + F(4)	5 + F(5)	1 + F(7)
9	11 + F(3)	7 + F(5)	5 + F(6)	1 + F(8)
10	11 + F(4)	7 + F(6)	5 + F(7)	1 + F(9)
11	11 + F(5)	7 + F(7)	5 + F(8)	1 + F(10)
12	11 + F(6)	7 + F(8)	5 + F(9)	1 + F(11)
13	11 + F(7)	7 + F(9)	5 + F(10)	1 + F(12)
14	11 + F(8)	7 + F(10)	5 + F(11)	1 + F(13)
15	11 + F(9)	7 + F(11)	5 + F(12)	1 + F(14)

Table 3.4 contains problems and their corresponding maximum values with items that add to the sub-problems. The first column contains the subset of the weight of the knapsack. The first row of Columns 2 to 4 contain items that are included in the knapsack to get maximum value, whiles the remaining rows contain the maximum value that can be attained by adding the corresponding item. Column 5 contains the maximum value for each knapsack sub-problem. Column 6 contains the item that contributed to that maximum value in column 5. We will lookup this table for solution to sub-problems and the items that gave those solutions.

Table 3.4 Knapsack Sub-problems solutions

Knapsack Sub-problems	Item weight and maximum values 6	4	3	1	F(s)	Item Label for F(s)
0	0	0	0	0	0	
1	-∞	-∞	-∞	1	1	1
2	-∞	-∞	-∞	2	2	1
3	-∞	-∞	5	3	5	3
4	-∞	7	6	6	7	4
5	-∞	8	7	8	8	4
6	11	9	10	9	11	6
7	12	12	12	12	12	6
8	13	14	13	13	14	4
9	16	15	16	15	16	6
10	18	18	17	17	18	6
11	19	19	19	19	19	6
12	22	21	21	20	22	6
13	23	23	23	23	23	6
14	25	25	24	24	25	6
15	27	26	27	26	27	6

Maximum for F(15) is 27 (Table 3.4). From Table 3.3 sub-problem that gave this maximum is F(9). F(9) also has a maximum of 16 with sub-problem F(3). F(3) has maximum 5 with sub-problem F(0). Therefore, from Table 3.4 items that can be included in the knapsack to maximize the value are 6, 6, 3. Therefore optimal value = 27 and corresponding solution is (2, 0, 1, 0)

3.5.4 The Branch and Bound Method

Branch and Bound is a class of exact algorithms for various optimization problems, especially integer programming problems and combinatorial optimization problems (COP). It partitions the solution space into smaller sub-problems that can be solved independently (branching).Bounding

discards sub-problems that cannot contain the optimal solution, thus decreasing the size of thesolution space. Given a maximization problem

a. Branch and Bound algorithm iteratively partitions the solution space S.
b. For each sub-problem an upper bound on the objective value is calculated.
c. When a feasible solution (i.e., no fractional variables remaining) is found, all sub-problems whose upper bounds are lower than this solution's objective value can be discarded.
d. The best known feasible solution represents a lower bound for all sub-problems, and only sub-problems with an upper bound greater than the global lower bound have to be considered.

3.5.5 Rounding Fractional Solutions.

After solving the LP relaxation we have an LP optimal z_{LP} which is a lower bound to the integer optimum z_{IP} . We use rounding-up procedures to get the integer optimum. A simple and useful heuristic is 'Largest In Least Empty' (LILE). This heuristicis described in the following four steps.

1. Round the fractional solution values downwards, and determine the unmetdemand.

2. Sort the finals in the unmet demand from largest to smallest.

3. Place the largest final from the unmet demand in the least empty rawthat can contain this final. If this is not possible, an extra raw must beadded.

4. Continue this process until the sorted list of finals from the unmet demandis completely allocated.

It is possible that a pattern generated by this algorithm is one of the patternsused in the relaxed integer programming solution. The LILE algorithm tends to minimize the numberof extra stocks required, and turns out to work quite well in practice.

3.5.6 An Illustrative Example

We have a demand to cut 511 pieces of 9-inch wood, 301 pieces of 8-inch wood, 263 pieces of 7-inch wood and 383 pieces of 6-inch wood from a given stock of 20-inch wood. Cost of stock is Gh¢ 1. We assume an unlimited quantity of stock is available. Table 3.5 summarizes this instance. The first column of Table 3.5 shows the smaller lengths of rod that are ordered by a customer and column two indicates the quantity of such length ordered.

Table 3.5 Order demands with quantity

Requested length(inch)	**Order Quantity**
9	511
8	301
7	263
6	383

Solution:

Number of itemtypes, m = 4

Stock Length = 20 inches.

Cost of stock length, c = Gh¢ 1

B = basis matrix (m x m)

B^{-1} = Inverse of the basis matrix.

Y = vector of dual variables,

Z = total cost of stocks length used

$l_1 = 9, l_2$=8, $l_3 = 7, l_4 = 6$

$L = 20$

RHS= vector of values on the right-hand-side of constraints.

Our initial m patterns will be such that pattern i cuts only the ith item type, i.e.

$$a_{ii} = \left\lfloor \frac{L}{l_i} \right\rfloor, and$$

$$a_{ik} = 0, i \neq k,$$

$$k = 1, \dots, 4 \; and \; \forall \; i = 1, \dots, 4$$

$\lfloor x \rfloor$ =largest integer less than x

Pattern 1 gives$a_{11} = \lfloor 20/9 \rfloor = 2, a_{21} = 0, a_{31} = 0, a_{41} = 0$ and Pattern 1= (2, 0, 0, 0)

Pattern 2 gives$a_{12} = 0, a_{22} = \lfloor 20/8 \rfloor = 2, a_{32} = 0, a_{42} = 0$ and Pattern 2= (0, 2, 0, 0)

Pattern 3 gives$a_{13} = 0, a_{23} = 0, a_{33} = \lfloor 20/7 \rfloor = 2, a_{43} = 0$ and Pattern 3= (0, 0, 2, 0)

Pattern 4 gives$a_{14} = 0, a_{24} = 0, a_{34} = 0, a_{44} = \lfloor 20/6 \rfloor = 3$ and Pattern 4= (0, 0, 0, 3)

Our initial set of patterns is given as

Pat1	Pat2	Pat3	Pat4
2	0	0	0
0	2	0	0
0	0	2	0
0	0	0	3

These four patterns will form our demand constraint set and also our initial basis matrix.

A formulation of this problem will be

$$Min\ c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4$$

subject to

$$2x_1 + 0x_2 + 0x_3 + 0x_4 \geq 511$$

$$0x_1 + 2x_2 + 0x_3 + 0x_4 \geq 301$$

$$0x_1 + 0x_2 + 2x_3 + 0x_4 \geq 263$$

$$0x_1 + 0x_2 + 0x_3 + 3x_4 \geq 383$$

$$x_1, x_2, x_3, x_4 \geq 0$$

Since the cost of the stock from which pattern i is cut, $c_i$equals 1, the objective function can be written as

$$Min\ x_1 + x_2 + x_3 + x_4$$

Basis (B) = $\begin{vmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \end{vmatrix}$ $C_B = [1\ 1\ 1\ 1]$

B^{-1}= $\begin{vmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0.333 \end{vmatrix}$

$$RHS = \begin{bmatrix} 511 \\ 301 \\ 263 \\ 383 \end{bmatrix}$$

$$\text{X} = B^{-1} \times RHS = \times \begin{bmatrix} 51 \\ 30 \\ 26 \\ 38 \end{bmatrix} \left| \begin{matrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0.333 \end{matrix} \right|$$

$x_1 = 255.5, x_2 = 150.5, x_3 = 131.5, x_4 = 127.67$

$Z = C_B \cdot X = 1 \times 255.5 + 1 \times 150.5 + 1 \times 131.5 + 1 \times 127.67 = 665.167$

Iteration 1

1. Compute dual prices

$$Y = C_B B^{-1} = [1\ 1\ 1\ 1] \left| \begin{matrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0.333 \end{matrix} \right|$$

$Y = [0.5, 0.5, 0.5, 0.3333]$

2. Generate entering column

Let pattern $P = (a, b, c, d)^T$ such that

$$YP > 1$$

$$and \quad 9a + 8b + 7c + 6d \leq 20$$

by dynamic programming

$F(s) = max\{y_i + F(s - l_i)\}$

s={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20}

$F(0) = 0, F(-s) = -\infty$

$$F(20) = max\{0.5 + F(20-9), 0.5 + F(20-8), 0.5 + F(20-7), 0.333 + F(20-6)\}$$

$$= max\{0.5 + F(11), 0.5 + F(12), 0.5 + F(13), 0.333 + F(14)\}$$

Therefore to find F (20) we have to find the maximum among sub-problems F(11), F(12), F(13) and F(14). These problems will also have sub-problems. Table 3.5 shows knapsack problems and their sub-problems. The sub-problems that contributed to the maximum are in the boxes.

Table 3.5 Knapsack problems and their sub-problems.

Knapsack	Item lengths and sub-problem			
Sub-problems	9	8	7	6
0	0	0	0	0
1	0.5 + F(-8)	0.5 + F(-7)	0.5 + F(-6)	0.3333 + F(-5)
2	0.5 + F(-7)	0.5 + F(-6)	0.5 + F(-5)	0.3333 + F(-4)
3	0.5 + F(-6)	0.5 + F(-5)	0.5 + F(-4)	0.3333 + F(-3)
4	0.5 + F(-5)	0.5 + F(-4)	0.5 + F(-3)	0.3333 + F(-2)
5	0.5 + F(-4)	0.5 + F(-3)	0.5 + F(-2)	0.3333 + F(-1)
6	0.5 + F(-3)	0.5 + F(-2)	0.5 + F(-1)	0.3333 + F(0)
7	0.5 + F(-2)	0.5 + F(-1)	0.5 + F(0)	0.3333 + F(1)
8	0.5 + F(-1)	0.5 + F(0)	0.5 + F(1)	0.3333 + F(2)
9	0.5 + F(0)	0.5 + F(1)	0.5 + F(2)	0.3333 + F(3)
10	0.5 + F(1)	0.5 + F(2)	0.5 + F(3)	0.3333 + F(4)
11	0.5 + F(2)	0.5 + F(3)	0.5 + F(4)	0.3333 + F(5)
12	0.5 + F(3)	0.5 + F(4)	0.5 + F(5)	0.3333 + F(6)
13	0.5 + F(4)	0.5 + F(5)	0.5 + F(6)	0.3333 + F(7)
14	0.5 + F(5)	0.5 + F(6)	0.5 + F(7)	0.3333 + F(8)
15	0.5 + F(6)	0.5 + F(7)	0.5 + F(8)	0.3333 + F(9)
16	0.5 + F(7)	0.5 + F(8)	0.5 + F(9)	0.3333 + F(10)
17	0.5 + F(8)	0.5 + F(9)	0.5 + F(10)	0.3333 + F(11)
18	0.5 + F(9)	0.5 + F(10)	0.5 + F(11)	0.3333 + F(12)
19	0.5 + F(10)	0.5 + F(11)	0.5 + F(12)	0.3333 + F(13)
20	0.5 + F(11)	0.5 + F(12)	0.5 + F(13)	0.3333 + F(14)

Table 3.6 shows the evaluated values for the sub-problems. The values in the box show the maximum among the values and the item length that was included to give this maximum.

Table 3.6 Knapsack sub-problems and their solutions

Knapsack Sub-problems	Item lengths and sub-problem solutions 9	8	7	6	F(s)	Item Label for F(s)
0	0	0	0	0	0	
1	-∞	-∞	-∞	-∞	-∞	
2	-∞	-∞	-∞	-∞	-∞	
3	-∞	-∞	-∞	-∞	-∞	
4	-∞	-∞	-∞	-∞	-∞	
5	-∞	-∞	-∞	-∞	-∞	
6	-∞	-∞	-∞	0.33333	0.33333	6
7	-∞	-∞	0.5	-∞	0.5	7
8	-∞	0.5	-∞	-∞	0.5	8
9	0.5	-∞	-∞	-∞	0.5	9
10	-∞	-∞	-∞	-∞	-∞	
11	-∞	-∞	-∞	-∞	-∞	
12	-∞	-∞	-∞	0.66667	0.66667	6
13	-∞	-∞	0.83333	0.83333	0.83333	7
14	-∞	0.83333	1	0.83333	1	7
15	0.83333	1	1	0.83333	1	8
16	1	1	1	-∞	1	9
17	1	1	-∞	-∞	1	9
18	1	-∞	-∞	1	1	9
19	-∞	-∞	1.16667	1.16667	1.16667	7
20	-∞	1.16667	1.33333	1.33333	1.33333	7

From Table 3.6 maximum for F(20) is 1.33333. From Table 3.5 sub-problem that gave this maximum is F(13). F(13) also has a maximum of 0.83333 with sub-problem F(6). F(6) has maximum 0.33333 with sub-problem F(0). Therefore, from Table 3.6 items that can be included in the knapsack to maximize the value are 7, 7, 6. Therefore optimal value = 1.33333 and corresponding Pattern is (0, 0, 2, 1).The entering pattern $P = (0, 0, 2, 1)^T$

3. Update new found column

$$\bar{P} = P \cdot B^{-1}$$

$$\bar{P} = P \cdot B^{-1} = \begin{vmatrix} 0 \\ 0 \\ 2 \\ 1 \end{vmatrix} \begin{vmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0.333 \end{vmatrix} \begin{vmatrix} 0 \\ 0 \\ 1 \\ 0.333 \end{vmatrix}$$

4. Compute leaving variable by ratio test.

Divide X by $\bar{P}$ component-wise where applicable.

131.5/1=131.5, 127.67/0.333=383.2733

Since 131.5 < 383.2733 and corresponds to the 3rd component, the 3rd column is the leaving variable.

we update the Basis B by replacingthe leaving variable with the entering variable and find the inverse to getB^{-1}.

$$B = \begin{vmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 1 & 3 \end{vmatrix}$$

$$B^{-1} = \begin{vmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & -0.1667 \\ 0 & 0 & 0 & 0.333 \end{vmatrix}$$

5. Update Solution

$$X = B^{-1} \times X = \begin{vmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & -0.1667 \\ 0 & 0 & 0 & 0.333 \end{vmatrix} \begin{vmatrix} 255.5 \\ 150.5 \\ 131.5 \\ 127.67 \end{vmatrix} = \begin{vmatrix} 255.5 \\ 150.5 \\ 131.5 \\ 83.833 \end{vmatrix}$$

$$Z = C_B \cdot X = 1 \times 255.5 + 1 \times 150.5 + 1 \times 131.5 + 1 \times 83.833 = 621.333$$

Iteration 2

1. Compute dual prices

$$Y = C_B B^{-1} = [1\ 1\ 1\ 1] \begin{vmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & -0.1667 \\ 0 & 0 & 0 & 0.333 \end{vmatrix}$$

$$Y = [0.5, 0.5, 0.3333, 0.3333]$$

2. Generate entering column

Let pattern $P = (a, b, c, d)^T$ such that

$$YP > 1$$

$$and \quad 9a + 8b + 7c + 6d \leq 20$$

by dynamic programming

$$F(s) = max\{y_i + F(s - l_i)\}$$

s={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20}

$$F(0) = 0, F(-s) = -\infty$$

$$F(20) = max\{0.5 + F(20-9), 0.5 + F(20-8), 0.3333 + F(20-7), 0.3333 + F(20-6)\}$$

$$= max\{0.5 + F(11), 0.5 + F(12), 0.3333 + F(13), 0.3333 + F(14)\}$$

To find F(20) we have to find the maximum among sub-problems F(11), F(12), F(13) and F(14). These problems will also have sub-problems. Table 3.7 shows the knapsack problems and their sub-problems. The sub-problems that contributed to the maximum are in the boxes

Table 3.7 Knapsack problems and their sub-problems

Knapsack	Item lengths and sub-problem			
Sub-problems	9	8	7	6
0	0	0	0	0
1	0.5 + F(-8)	0.5 + F(-7)	0.3333+ F(-6)	0.3333 + F(-5)
2	0.5 + F(-7)	0.5 + F(-6)	0.3333+ F(-5)	0.3333 + F(-4)
3	0.5 + F(-6)	0.5 + F(-5)	0.3333+ F(-4)	0.3333 + F(-3)
4	0.5 + F(-5)	0.5 + F(-4)	0.3333+ F(-3)	0.3333 + F(-2)
5	0.5 + F(-4)	0.5 + F(-3)	0.3333+ F(-2)	0.3333 + F(-1)
6	0.5 + F(-3)	0.5 + F(-2)	0.3333+ F(-1)	0.3333 + F(0)
7	0.5 + F(-2)	0.5 + F(-1)	0.3333+ F(0)	0.3333 + F(1)
8	0.5 + F(-1)	0.5 + F(0)	0.3333+ F(1)	0.3333 + F(2)
9	0.5 + F(0)	0.5 + F(1)	0.3333 + F(2)	0.3333 + F(3)
10	0.5 + F(1)	0.5 + F(2)	0.3333+ F(3)	0.3333 + F(4)
11	0.5 + F(2)	0.5 + F(3)	0.3333+ F(4)	0.3333 + F(5)
12	0.5 + F(3)	0.5 + F(4)	0.3333 + F(5)	0.3333 + F(6)
13	0.5 + F(4)	0.5 + F(5)	0.3333+ F(6)	0.3333 + F(7)
14	0.5 + F(5)	0.5 + F(6)	0.3333+ F(7)	0.3333 + F(8)
15	0.5 + F(6)	0.5 + F(7)	0.3333 + F(8)	0.3333 + F(9)
16	0.5 + F(7)	0.5 + F(8)	0.3333+ F(9)	0.3333 + F(10)
17	0.5 + F(8)	0.5 + F(9)	0.3333+ F(10)	0.3333 + F(11)
18	0.5 + F(9)	0.5 + F(10)	0.3333+ F(11)	0.3333 + F(12)
19	0.5 + F(10)	0.5 + F(11)	0.3333+ F(12)	0.3333 + F(13)
20	0.5 + F(11)	0.5 + F(12)	0.3333+ F(13)	0.3333 + F(14)

Table 3.8 shows the evaluated values for the sub-problems. The values in the box show the maximum among the values and the item length that was included to give this maximum.

Table 3.8: Knapsack sub-problems and their solutions

Knapsack Sub-problems	Item weight and sub-problems solutions 9	8	7	6	F(s)	Item Label for F(s)
0	0	0	0	0	0	
1	-∞	-∞	-∞	-∞	-∞	
2	-∞	-∞	-∞	-∞	-∞	
3	-∞	-∞	-∞	-∞	-∞	
4	-∞	-∞	-∞	-∞	-∞	
5	-∞	-∞	-∞	-∞	-∞	
6	-∞	-∞	-∞	0.33333	0.33333	6
7	-∞	-∞	0.33333	-∞	0.33333	7
8	-∞	0.5	-∞	-∞	0.5	8
9	0.5	-∞	-∞	-∞	0.5	9
10	-∞	-∞	-∞	-∞	-∞	
11	-∞	-∞	-∞	-∞	-∞	
12	-∞	-∞	-∞	0.66667	0.66667	6
13	-∞	-∞	0.66667	0.66667	0.66667	7
14	-∞	0.83333	0.66667	0.83333	0.83333	8
15	0.83333	0.83333	0.83333	0.83333	0.83333	9
16	1	1	1	-∞	1	8
17	1	1	-∞	-∞	1	9
18	1	-∞	-∞	1	1	9
19	-∞	-∞	1	1	1	7
20	-∞	1.16667	1	1.16667	1.16667	8

Maximum for F(20) is 1.16667 (Table 3.8). From Table 3.7 sub-problem that gave this maximum is F(12). F(12) also has a maximum of 0.66667 with sub-problem F(6). F(6) has maximum 0.33333 with sub-problem F(0). Therefore, from Table 3.8 items that can be included in the knapsack to maximize the value are 8, 6, 6.

Therefore optimal value = 1.16667 and corresponding Pattern is (0, 1, 0, 2).

The entering pattern $P = (0, 1, 0, 2)^T$

3. Update new found column

$$\bar{P} = P \cdot B^{-1}$$

$$\bar{P} = P \cdot B^{-1} = \begin{vmatrix} 0 \\ 1 \\ 0 \\ 2 \end{vmatrix} \begin{vmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & -0.166 \\ 0 & 0 & 0 & 0.333 \end{vmatrix} \begin{vmatrix} 0 \\ 0.5 \\ 0 \\ 0.666 \end{vmatrix}$$

4. Compute leaving variable by ratio test.

Divide N by $\bar{P}$ component-wise where applicable.

150.5/0.5=301, 83.833/0.666=125.875

Since 127.875<301 and corresponds to the 4th component, the 4th column is the leaving variable.

we update the new Basis B and its inverse B^{-1}.

$$B = \begin{vmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 1 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 1 & 2 \end{vmatrix}$$

$$B^{-1} = \begin{vmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0.125 & 0.5 & -0.25 \\ 0 & -0.25 & 0 & 0.5 \end{vmatrix}$$

5. Update Solution

$$X = B^{-1} \times X = \begin{vmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0.125 & 0.5 & -0.25 \\ 0 & -0.25 & 0 & 0.5 \end{vmatrix} \begin{vmatrix} 255.5 \\ 150.5 \\ 131.5 \\ 83.833 \end{vmatrix} = \begin{vmatrix} 255.5 \\ 87.625 \\ 131.5 \\ 125.75 \end{vmatrix}$$

$$Z = C_B \cdot X = 1 \times 255.5 + 1 \times 87.625 + 1 \times 131.5 + 1 \times 125.75 = 600.375$$

Iteration 3

1. Compute dual prices

$$Y = C_B B^{-1} = [1\ 1\ 1\ 1] \begin{vmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0.125 & 0.5 & -0.25 \\ 0 & -0.25 & 0 & 0.5 \end{vmatrix}$$

$$Y = [0.5, 0.5, 0.375, 0.25]$$

2. Generate entering column

Let pattern $P = (a, b, c, d)^T$ such that

$$YP > 1$$

$$and \quad 9a + 8b + 7c + 6d \leq 20$$

using dynamic programming

$$F(s) = max\{y_i + F(s - l_i)\}$$

s={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20}

$$F(0) = 0, F(-s) = -\infty$$

$$F(20) = max\{0.5 + F(20-9), 0.5 + F(20-8), 0.375 + F(20-7), 0.25 + F(20-6)\}$$

$$= max\{0.5 + F(11), 0.5 + F(12), 0.375 + F(13), 0.25 + F(14)\}$$

Therefore to find F(20) we have to first find sub-problems F(11), F(12), F(13) and F(14). These problems will also have sub-problems. Table 3.9 shows the knapsack problems and their sub-problems. The sub-problems that contributed to the maximum are in the boxes

Table 3.9 Knapsack problems and their sub-problems

Knapsack	Items weights			
Sub-problems	9	8	7	6
0	0	0	0	0
1	0.5 + F(-8)	0.5 + F(-7)	0.375+ F(-6)	0.25 + F(-5)
2	0.5 + F(-7)	0.5 + F(-6)	0.375+ F(-5)	0.25 + F(-4)
3	0.5 + F(-6)	0.5 + F(-5)	0.375+ F(-4)	0.25 + F(-3)
4	0.5 + F(-5)	0.5 + F(-4)	0.375+ F(-3)	0.25 + F(-2)
5	0.5 + F(-4)	0.5 + F(-3)	0.375+ F(-2)	0.25 + F(-1)
6	0.5 + F(-3)	0.5 + F(-2)	0.375+ F(-1)	0.25 + F(0)
7	0.5 + F(-2)	0.5 + F(-1)	0.375+ F(0)	0.25 + F(1)
8	0.5 + F(-1)	0.5 + F(0)	0.375+ F(1)	0.25 + F(2)
9	0.5 + F(0)	0.5 + F(1)	0.375 + F(2)	0.25 + F(3)
10	0.5 + F(1)	0.5 + F(2)	0.375+ F(3)	0.25 + F(4)
11	0.5 + F(2)	0.5 + F(3)	0.375+ F(4)	0.25 + F(5)
12	0.5 + F(3)	0.5 + F(4)	0.375 + F(5)	0.25 + F(6)
13	0.5 + F(4)	0.5 + F(5)	0.375+ F(6)	0.25 + F(7)
14	0.5 + F(5)	0.5 + F(6)	0.375+ F(7)	0.25 + F(8)
15	0.5 + F(6)	0.5 + F(7)	0.375 + F(8)	0.25 + F(9)
16	0.5 + F(7)	0.5 + F(8)	0.375+ F(9)	0.25 + F(10)
17	0.5 + F(8)	0.5 + F(9)	0.375+ F(10)	0.25 + F(11)
18	0.5 + F(9)	0.5 + F(10)	0.375+ F(11)	0.25 + F(12)
19	0.5 + F(10)	0.5 + F(11)	0.375+ F(12)	0.25 + F(13)
20	0.5 + F(11)	0.5 + F(12)	0.375+ F(13)	0.25 + F(14)

Table 3.10 shows the evaluated values for the sub-problems. The values in the box show the maximum among the values and the item length that was included to give this maximum

Table3.10 Knapsack sub-problems and their solutions

Knapsack Sub-problems	Item weight and maximum values 9	8	7	6	F(s)	Item Label for F(s)
0	0	0	0	0	0	
1	-∞	-∞	-∞	-∞	-∞	
2	-∞	-∞	-∞	-∞	-∞	
3	-∞	-∞	-∞	-∞	-∞	
4	-∞	-∞	-∞	-∞	-∞	
5	-∞	-∞	-∞	-∞	-∞	
6	-∞	-∞	-∞	0.25	0.25	6
7	-∞	-∞	0.375	-∞	0.375	7
8	-∞	0.5	-∞	-∞	0.5	8
9	0.5	-∞	-∞	-∞	0.5	9
10	-∞	-∞	-∞	-∞	-∞	
11	-∞	-∞	-∞	-∞	-∞	
12	-∞	-∞	-∞	0.5	0.5	6
13	-∞	-∞	0.625	0.625	0.625	7
14	-∞	0.75	0.75	0.75	0.75	8
15	0.75	0.875	0.875	0.75	0.875	9
16	0.875	1	0.875	-∞	1	8
17	1	1	-∞	-∞	1	9
18	1	-∞	-∞	1	1	9
19	-∞	-∞	0.875	0.875	0.875	7
20	-∞	1	1	1	1	8

Maximum for F(20) is 1 (Table 3.8). From Table 3.7 sub-problem that gave this maximum is F(12). F(12) also has a maximum of 0.5 with sub-problem F(6). F(6) has maximum 0.25 with sub-problem F(0). Therefore, from Table 3.8 items that can be included in the knapsack to maximize the value are 8, 6, 6.

Therefore optimal value = 1 and corresponding Pattern is (0, 1, 0, 2).

This pattern has already been generated in iteration 2.

Therefore the process terminates and the solution is as follows:

The Optimal Fractional Solution is given as

Size	Pat1	Pat2	Pat3	Pat4
9	2	0	0	0
8	0	2	0	1
7	0	0	2	0
6	0	0	1	2
Qty	255.5	87.625	131.5	125.75

Cost of the total stock used is 600.375 and the number of Iterations is 3

2 pieces of 9-inch is cut in Pattern 1,

2 x 255.5 = 511pieces of 9-inch.

2 pieces of 8-inch is cut in Pattern 2 and 1 piece in Pattern4,

2 x 87.625+ 1 x 125.75= 301pieces of 8-inch

2 pieces of 7-inch is cut in Pattern 3

2 x 131.5 = 263 pieces of 7-inch.

1 piece of 6-inch is cut in Pattern 3 and 2 pieces in Pattern 4,

1 x 131.5 + 2 x 125.75 = 383.

and the quantities of stock used = 255.5 + 87.625 + 131.5 + 125.75 = 600.375

We round the above solution to their upper integer values, since we will use discrete quantities of stock.

$\lceil 255.5 \rceil = 256 pieces$

$\lceil 87.625 \rceil = 88 pieces$

$\lceil 131.5 \rceil = 132 pieces$

$\lceil 125.75 \rceil = 126 pieces$

where $\lceil x \rceil$ represents the upper integer value of x.

The integer solution gives

2 pieces of 9-inch is cut in Pattern 1,

2 x 256 = 512 pieces of 9-inch.

2 pieces of 8-inch is cut in Pattern 2 and 1 piece in Pattern4,

2 x 88 + 1 x 126= 302 pieces of 8-inch

2 pieces of 7-inch is cut in Pattern 3

2 x 132 = 264 pieces of 7-inch.

1 piece of 6-inch is cut in Pattern 3 and 2 pieces in Pattern 4,

1 x 132 + 2 x 126 = 384.

and the quantities of stock used = 256 + 88 + 132 + 126 = 602

The integer values show that

512 pieces of 9-inch was cut which is ≥ 511, and satisfy our demand constraint (1)

302 pieces of 8-inch was cut which is ≥ 302, and satisfy our demand constraint (2)

264 pieces of 7-inch was cut which is ≥ 263, and satisfy our demand constraint (3)

384 pieces of 6-inch was cut which is ≥ 383, and satisfy our demand constraint (4)

Total stock lengths used is 602

Total cost of stock lengths used = 1 x 602 = GH¢ 602.

CHAPTER 4

DATA COLLECTION AND ANALYSIS

4.2 Data Collection

The data was collected from Rogersco Saw Mill Ltd, Yawkwei – Asante . The data includes the cross-sectional dimension of different sizes of strip boards, quantities of each strip board type and the total cost of the strip boards utilized to meet demand for the smaller sizes, cross-sectional dimension of smaller sizes requested and their respective quantities per order and the sample orders for August 2010 to December 2010. The cross-sectional width for both the strip boards and the cuts are 2-inch long and therefore are not shown. Example, cut-piece of 6 inches indicates a cut-piece of cross-sectional dimension 2 x 6. Likewise, a strip board of length 20 indicates a strip board of cross-sectional dimension 2 x 20.

Table 4.1 shows cross-sectional length of the cut-pieces from the company together with demand for each in sample orders made to the company. The first column indicates the cross-sectional length of the cut-pieces. The second to the last columns indicate quantities for the various cut-pieces ordered by customers.

Table 4.1 Order data from the company from August to December 2010

	Order Quantities				
Cuts-piece (inch)	August	September	October	November	December
6”	1000	540	1500	200	220
4”	500	850	200	150	115
3”	119	40	0	100	120
2”	203	85	203	120	40

Table 4.2 shows cross-sectional length of strip boards that are cut from these timber logs through the ripping process. It indicates the cross-sectional length of the strip board in inches in the first column and the cost of each strip board in the second column.

Table 4.2 Cross-sectional length of Strip after the Ripping process.

Stock Sizes (L-inches)	Cost (GHc)
20”	13.50
19”	13.00
17”	11.00

Table 4.3 shows the quantities of strip board types the company used in meeting the corresponding order demands in Table 4.1 and their cost.

Table 4.3 Company Strip quantities and cost utilized to meet the orders in table 4.1

Strip Length (inch)	Orders									
	August		September		October		November		December	
	Qty	Cost	Qty	Cost	Qty	Cost	Qty	Cost	Qty	Cost
20”	342	4,617	300	4,050	209	2,821.5	32	432	20	270
19”	0	0	20	260	167	2,171	40	520	33	429
17”	130	1,430	50	550	196	2,156	63	693	78	858
TotalCost		6,047		4,860		7,148.5		1,645		1,557

The company used 342 pieces of 20” and 130 pieces of 17” strip for the August order. Likewise, it used 300 pieces of 20”, 20 pieces of 19” and 50 pieces of 17” for the September order, 209 pieces of 20”, 167 pieces of 19” and 196 pieces of 17” for the October order. Furthermore, 32 pieces of 20”, 40 pieces of 19” and 63 pieces of 17” where used for the November order and 20 pieces of 20”, 33 pieces of 19” and 78 pieces for the December order. The total cost of strip board are GH6,047, GH4,860, GH7,148.50, GH1,645, GH1,557 for the corresponding August order, September order, October order, November order and the December order respectively.

4.3 Problem Formulation

The problem is to find, for each order, the minimum total cost of stock lengths utilized in cutting at least the total quantities demanded of each cut-piece. The problem is formulated as a multi-

objective function of a linear program anda knapsack problem with the assumption that quantities available of each strip board type is unlimited.

The notation and formulation are as follows. Let

m:number of cut-piece type. This is also the number of constraints.

n:number of patterns or variables.

k:number of different strip lengths.

l_i: length of cut piece type $i, i = 1, \dots, m$.

L_j:length of strip $j, j = 1, \dots, k$.

j:the*j-th*pattern.

x_j:number of times pattern *j* is used in the solution. These are the decision variables for the LP

a_{ij}:number of cut-piece *i* in pattern *j*. This is the coefficient in the demand constraint.

b_i:the demand for cut-piece *i* for each order

c_j:cost of the strip used by pattern*j*

y_j:dual cost of pattern *j*

Z:total cost of strips used

P: $(p_1, \dots, p_m) be$a feasible pattern that cuts from strip L_j with cost c_j

The linear programming problem is

$$Min\ Z = \sum_{j}^{n} c_j x_j \qquad (4.1)$$

subject to

$$\sum_{j=1}^{n} a_{ij}x_j \geq \quad \forall\, i = 1, \dots, m \qquad (4.2)$$

$$x_j \geq 0 \qquad (4.3)$$

The c_j is from column 2 of Table 4.2 and b_i is from Table 4.1.

The dual is given as

$$Max \sum_{i=1}^{m} y_i b_i \qquad (4.4)$$

$$s.t \sum_{i=1}^{m} a_{ij}y_j \leq c_j, \qquad \forall\, j = 1, \dots, n \qquad (4.5)$$

$$y_i \geq 0 \quad (4.6)$$

The dual variables are calculated as

$$y = C_B B^{-1}$$

where B^{-1} is the inverse of the basis matrix and C_B is the cost vector corresponding to the columns of the basis matrix.

The column generation knapsack problem is to find a feasible pattern $P = (p_1, \dots, p_m)$ that cuts from strip L_j with cost c_j such that

$$yP > c_j \quad (4.7)$$

We also determine the feasibility of the pattern as

$$\sum_{i=1}^{m} l_i p_i \leq L_j \;\; and \;\; p_i \geq 0 \; and \; integer \quad (4.8)$$

The l_i is from column 1 of Table 4.1 and L_j is column 1 of Table 4.2

The two inequalities (4.7) and (4.8) find a feasible pattern P that improved our current Linear programming solution.

Wecreated the *m* initial cutting patterns such that

$$a_{ii} = \left\lfloor \frac{L_j}{l_i} \right\rfloor$$

$$a_{ik} = 0, i \neq k$$

$$i = 1, \ldots, m \; and \; k = 1, \ldots, m.$$

$L_j$islargest of the strip in Table 4.2 for which $L_j > l_i$. The cost of $L_j$is c_j. The m initial patterns are as follows:

Pattern 1 is $a_{11} = \lfloor L_j/l_1 \rfloor, a_{21} = 0, \cdots, a_{m1} = 0$

Pattern 2 is $a_{12} = 0, a_{22} = \lfloor L_j/l_2 \rfloor, \cdots, a_{m2} = 0$

$\vdots$

Pattern m is $a_{1n} = 0, a_{2n} = 0, \cdots, a_{mn} = \lfloor L_j/l_m \rfloor$

Our initial m patterns therefore are

Pat1	Pat2		Pat m
a_{11}	0	$\cdots$	0
0	a_{22}	0	0
$\vdots$	$\vdots$	$\ddots$	$\vdots$
0	0	$\cdots$	a_{mn}

The Linear Programming problem is then given as

$$Min\, Z = c_1x_1 + c_2x_2 + \cdots + c_nx_n \qquad (4.9)$$

subject to

$$a_{11}x_1 + 0x_2 + \cdots + 0x_n \geq b_1$$

$$0x_1 + a_{22}x_2 + \cdots + 0x_n \geq b_2 \qquad (4.10)$$

$$\vdots$$

$$0x_1 + 0x_2 + \cdots + a_{mn}x_n \geq b_m$$

$$x_1, x_2, \cdots, x_n \geq 0 \qquad (4.11)$$

The objective function (4.9) is to minimize the total cost of stock lengths used. Constraint (4.10) requires that the quantity produced is at least equal to quantity demanded for each cut-piece i. Constraint (4.11) also shows that the initial basis matrix, B, is the initial m patterns. Constraint (3) is the non-negativity restriction on the decision variables.

The dual formulation is given as

$$Max\ Z = b_1 y_2 + b y_2 + \cdots + b_m y_m \quad (4.12)$$

subject to

$$\begin{aligned} a_{11} y_1 + 0 y_2 + \cdots + 0 y_m &\leq c_1 \\ 0 y_1 + a_{22} y_2 + \cdots + 0 y_m &\leq c_2 \qquad (4.13) \\ &\vdots \\ 0 y_1 + 0 y_2 + \cdots + a_{nm} y_m &\leq c_n \end{aligned}$$

$$y_1, y_2, \cdots, y_m \geq 0 \qquad (4.14)$$

4.4 Cutting Stock Algorithm

The steps for the solution of the cutting stock problem is given below

Step 1: Formulate the problem as a linear programming (LP) master problem.

Step 2: Solve for dual of the LP master problem using the simplex method.

Step 3:Use the dual solution in step 2 to solve the auxiliary knapsack sub-problem to find a feasible pattern with a negative reduced cost, i.e. $C_j - Z_j < 0$. If such a pattern exists, then update the LP master problem with the new pattern and go to step 2.

Step 4: If there is no such pattern in step 3, then the current LP solution is the optimal solution and we stop.

Step 5: Round the optimal LP solution to get the optimal integer program solution.

4.5 Computational Procedure and Results

A Visual Basic.Net code was developed to implement the above algorithm on an Intel Atom N455 computer processor with speed 1.66GHz and 2GB of RAM. Output from the application for the five orders is given in Table 4.4 – 4.10. Table 4.4 shows quantities of cut-pieces that were cut using the application for the five set of orders, August to December, 2010. The first column indicates the length of the cut-pieces. The second to the last columns indicate quantities for the cut-pieces produced by the application for each month.

Table 4.4 Quantities of cut-pieces generated by application for the five order demands for August to December, 2010.

	Quantities of cut-pieces				
Cuts L-inch	August	September	October	November	December
6”	1000	541	1501	200	220
4”	500	851	200	150	115
3”	119	40	-	100	120
2”	203	85	203	120	40

The “-“ in the table indicates that there was no order in the month of October for the 3” cut-piece length.

Table 4.5 shows the corresponding lengths and quantities of strip boards the application used in generating the output for the quantities in Table 4.4 and their associated cost. (See Appendix A for details of the application output)

Table 4.5 Quantity and cost of Strips used by application to generate quantities of Table 4.4.

Strip Length (inch)	Quantities and cost of Strips used									
	August		September		October		November		December	
	Qty	Cost	Qty	Cost	Qty	Cost	Qty	Cost	Qty	Cost
20”	329	4,441.5	313	4,225.5	202	2,740.5	32	432	-	-
19”	-	-	-	-	164	2,132	-	-	50	650
17”	129	1,419	40	440	200	2,200	100	1,100	75	825
Total Cost		5,860.5		4,665.5		7,072.5		1,532		1,475

The application used 329 pieces of 20” and 129 pieces of 17” strip for the August order. Likewise, it used 313 pieces of 20” and 40 pieces of 17” for the September order, 202 pieces of 20”, 164 pieces of 19” and 200 pieces of 17” for the October order. Furthermore, 32 pieces of 20” and 100 pieces of 17” where used for the November order and 50 pieces of 19” and 75 pieces for the December order. The total cost of strip board are GH5,860, GH4,665.5, GH7,072.50, GH1,532, GH1,475 for the August, September, October, November and the December orders respectively.

Table 4.6 – 4.10 represent the optimal cutting patterns that were generated, the number of times the pattern was used and the length of strip that was used by the pattern for each of the months. The first column shows the cut-piece length. The remaining columns show the patterns. Column 7 shows the strip length used, row 8 shows the number of times the pattern was used and the last row shows the number of iterations that were performed to get the optimal solution.

Table 4.6Optimal Pattern generated and their usage for the August orders

Cut-piece (l_i)	August (j)			
	Pattern 1	Pattern 2	Pattern 3	Pattern 4
6”	2	2	1	3
4”	1	2	2	0
3”	0	0	1	0
2”	0	0	0	1
Strip Used(L_j)	17”	20”	17”	20”
Pattern Usage (x_j)	10	126	119	203
Number of Iterations: 7				

The numbers in the table represents the quantity of the cut-pieces in each pattern. For example, pattern 1 was generated using 17” strip and each usage of the pattern produced 2 of 6”, 1 of 4” and none of the 3” and 2”. Since the pattern was used 10 times we have, for this pattern, 2x10=20 of 6” and 1x10=10 of 4”. The number of iterations was 7.

Table 4.7Optimal Pattern generated and their usage for the September orders

Cut-piece (l_i)	September (j)			
	Pattern 1	Pattern 2	Pattern 3	Pattern 4
6”	2	0	1	3
4”	2	5	2	0
3”	0	0	1	0
2”	0	0	0	1
Strip Used(L_j)	20”	20”	17”	20”
Pattern Usage(x_j)	123	105	40	85
Number of Iterations = 5				

Table 4.8Optimal Pattern generated and their usage for the October orders

Cut-piece (l_i)	October (j)		
	Pattern 1	Pattern 2	Pattern 3
6”	3	2	3
4”	0	1	0
2”	0	0	1
Strip Used(L_j)	19”	17”	20”
Pattern Usage (x_j)	164	200	203
Number of Iterations = 5			

Table 4.9Optimal Pattern generated and their usage for the November orders

Cut-piece (l_i)	November (j)			
	Pattern 1	Pattern 2	Pattern 3	Pattern 4
6”	3	1	2	0
4”	0	2	0	1
3”	0	1	1	1
2”	1	0	1	5
Strip Used(L_j)	20”	17”	17”	17”
Pattern Usage (x_j)	32	68	18	14
Number of Iterations = 6				

Table 4.10Optimal Pattern generated and their usage for the December orders

Cut-piece (l_i)	December(c) (j)			
	Pattern 1	Pattern 2	Pattern 3	Pattern 4
6”	3	1	2	2
4”	0	2	1	0
3”	0	1	1	1
2”	0	0	0	1
Strip Used(L_j)	19”	17”	19”	17”
Pattern Usage(x_j)	5	35	45	40
Number of Iterations = 8				

4.4 Discussion

A comparison of the output from the application and the data from the company in Table 4.3 shows an improvement over the ad-hoc method used by the company. Table 4.11 summarizes the total cost of strips used by the company and the application to meet the same set of demand. The first column indicates the month. The second and third columns show the total cost of strips for each month order by Company and application. The last column shows the difference in the total cost between the Company and the application.

Table 4.11 Summary of Total cost of strips used by Company and the application for set of orders.

Month (2010)	Total Cost of strips used to meet set of demands (GH ¢)		
	Company	Application	Difference
August	6,047.00	5,860.00	187.00
September	4,860.00	4,665.50	194.50
October	7,148.50	7,072.50	76.00
November	1,645.00	1,532.00	113.00
December	1,557.00	1,475.00	82.00
Total	21,257.50	20,604.50	652.50

The table shows that by the application of the techniques shown by this application, the company would have made a savings of GH187, GH195.5, GH76, GH113, and GH82 for the August, September, October, Novemberand the December orders respectively, a total savings of GH652.

4.5 Features of the Application

The software allows the user to input data into the application interface by typing data directly into Ordered Sizes grid (for size of cut-pieces) and the Stock Sizes grid (for size of stock lengths). We can also load data from text files by clicking the "load data" button. Results from the computation can also be saved to a text file by clicking the "Save Output) button.

(Codes attached: See Appendix B)

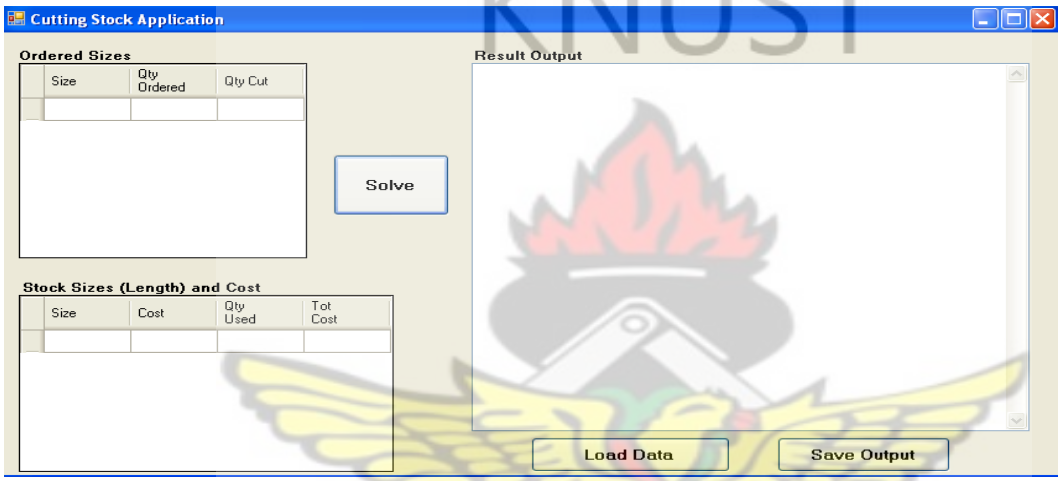

Fig 4.1 User interface for the Cutting stock application

The program generates an initial solution and shows both the fractional and the integer feasible solutions for the problem and selects the optimal solution together with the patterns that are needed to get the optimal solution.

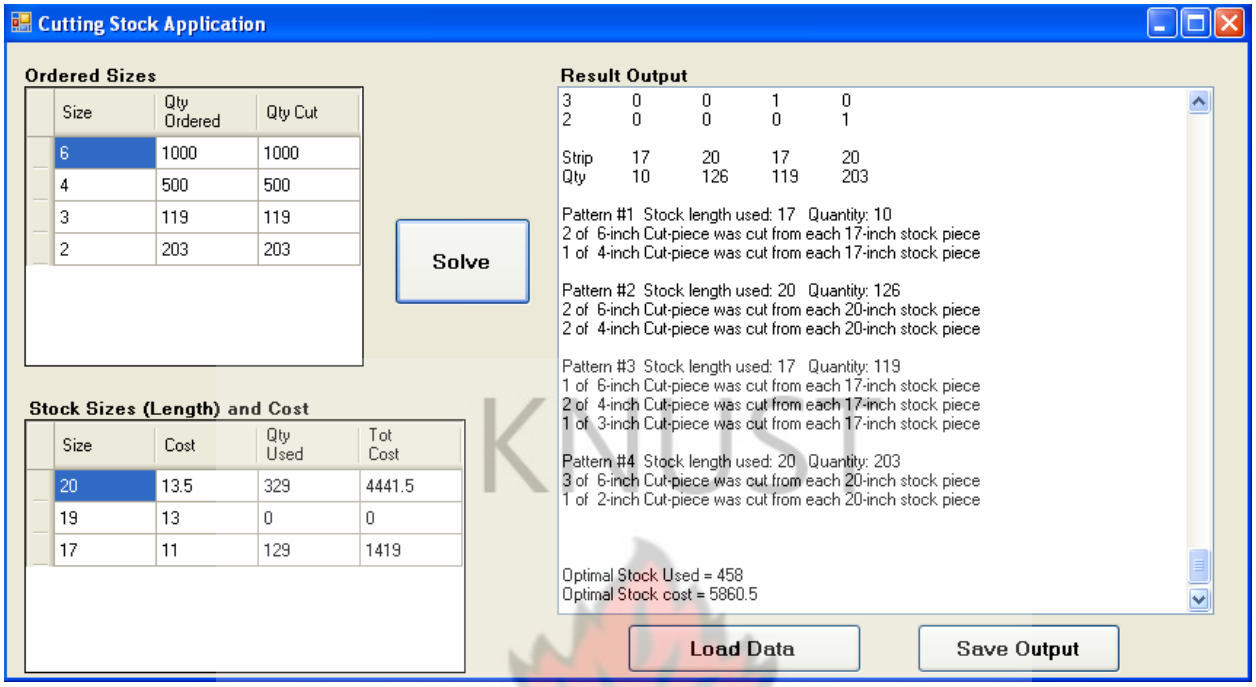

Fig 4.2 Output from the application

CHAPTER 5

CONCLUSION AND RECOMMENDATION

5.1 Conclusion

This thesis has modeled the cutting of strip boards as a one dimensional cutting stock problem and used linear programming techniques to find solution to it. Knapsack algorithm by dynamic programming was used to generate entering columns. The thesis generated optimal patterns (Tables 4.6 – 4.10) for the cutting of the strip boards that minimizes the total cost of strips utilized to meet demands for quantities of cut-pieces (Table 4.5). The study also shows that by the application of these techniques, total cost of stock lengths utilized can be minimized (Table 4.11). The thesis also developed a computer program that solves this cutting stock problem. The application can be used in any one dimensional cutting situation.

5.2 Recommendation

Rogersco Sawmill Limited should use the cutting stock procedure to generate cutting patterns for the cutting of all their strip boards. This will help to minimize the total cost of strip boards that are utilized to meet orders as indicated in Table 4.11. This consequently will lead to increase in profit and efficient utilization of timber. The use of the application is systematic and transparent as compared with the ad-hoc method used by the company.

The application can be used for any problem that can be modeled as a one-dimensional cutting stock problem. The thesis considered strip boards with fixed width, i.e. 2 inches, however future research should consider cases with variable widths. We also assumed unlimited quantities of strip boards. Future research should also consider instances where the quantities of stocks of strip boards become constraints.

REFERENCES

1. Amponsah, S.K. (2006), Optimization Techniques I, University Printing Press, KNUST, page 5.
2. Arbib, C. and Marinelli, F. (2007), An optimization model for trim loss minimization in an automotive glass plant, European Journal of Operational Research 183(3):1421-1432.
3. Belov G., Letchford A. and Uchoa E. A (2005), "Node-flow model for one dimensional stock cutting: Robust branch & cut & price," Niteroi: Technical Report RPEP, Vol. 5, No. 7, Universidade Federal Fluminense, Engenharia de Producao
4. Boschetti, M. A., Mingozzi A. (2003), The two-dimensional finite bin-packing problem. Part I: New lower bounds for the oriented case, 4OR 1 27–42.
5. Boschetti, M. A., Mingozzi A. (2003), The two-dimensional finite bin-packing problem. Part II: New lower and upper bounds, 4OR2 135–148.
6. Caprara, A. and Monaci M.(2004), On the two-dimensional knapsack problem, Operation. Research Letters32: 5–14.
7. Carvalho J. and Rodrigues A.(1995), A LP-based approach to a two-stage cutting stock problem. European Journal of Operational Research;84
8. Chen C. S. Hart S. M. and Tham W. (1996), A Simulated Annealing Heuristic for the One-dimensional Cutting Stock Problem., European Journal of Operational Research, 93(3): 522-535.
9. Christfides N. and Whitlock C. (1977), An algorithm for the two dimensional cutting problem, Journal of Operations Research 25: 30-44.
10. Dantzig G. B. (1957), Discrete ExtremumProblems, Operations Research 5: 161-310

11. Dell'Amico, M., Martello S. and Vigo D. (2002), A lower bound for the non-oriented two-dimensional bin-packing problem. Discrete Appl. Math.118: 13–24.
12. Dyckhoff, H. (1990), A typology of cutting an packing problems, European Journal of Operational Research 44: 145-159.
13. Dyson R. G. and Gregory A.S. (1974),The Cutting Stock Problem in the Flat Glass Industry, Operations Research Quarterly, Vol. 25, No. 1,pp. 41-53.
14. Eisemann K., The Trim Problem (1957), Management Science, Vol 3, No. 3, pp. 279-284
15. Farley A. (1990), The cutting stock problem in the canvas industry, European Journal of Operational Research 44, pp.247-255.
16. Fekete S. P. and Joerg S (2001), New classes of lower bounds for the bin packing problem. Math. Programming91, pp. 11–31.
17. Ferreira J. S., Neves M.A. and Castro P. F. (1990). A two-phase roll cutting problem. European Journal of Operational Research, pp. 44
18. Gilmore P. C. and Gomory R.E. (1961),A Linear Programming Approach to the Cutting Stock Problem: Part I, Vol. 9, No. 6 , pp. 849-859
19. Gilmore P.C. and Gomory R.E. (1963), A linear programming approach to the cutting stock problem: Part II, Operations Research 11, pp. 863 - 888
20. Gilmore P. C. and Gomory R.E. (1965), Multistage cutting stock problems of two and more dimensions. Operations Research 13, pp.94-120
21. Gilmore P.C. and Gomory R.E (1966),The theory and computation of knapsack functions. Operations Research 14, pp.1045-1074
22. Goulimis C. (1990), Optimal solutions for the cutting stock problem, European Journal of Operational Research 44: 197-208.

23. Hadjiconstantinou E. and Christofides N. (1995),An exact algorithm for general, orthogonal, two-dimensional knapsack problems,European Journal of Operational Research, 83 39–56.
24. Haessler R. W. and Sweeney P. E. (1991), Cutting stock problems and solution procedures, European Journal of Operational Research 54: p.141 – 150
25. Hahn S.G. (1968), On the Optimal Cutting of Defective Sheets, Operations Research 16, pp.1100-1113.
26. Hopper E. and Turton B. (1999),A Genetic Algorithm for a 2D Industrial Packing Problem, Computers & Industrial Engineering , 37: pp. 375-378.
27. Johnston R. E. and Sadinlija E. (2004), A new model for complete solutions to one-dimensional stock problems, European Journal of Operational Research,153, pp. 176-183.
28. KalvelagenE.(2002),Column Generation with GAMS, http://amsterdamoptimization.com/pdf/colgen.pdf
29. Lirov Y. (1992), Knowledge Based Approach to the Cutting Stock Problem., Mathematical and Computer Modeling, 16 (1), pp. 107-125.
30. Maculan N., Michelon P. and Plateau G.(1992), Column-generation in linear programming with bounding variable constraints and its application in integer programming. PesquisaOperacional, 12(2), pp. 45-57.
31. Matousek J. and Gartner B. (2007), Understanding and Using Linear Programming, pp. 9
32. Morabito R. and Arenales M. (2000), Optimizing the cutting of stock plates in a furniture company, International Journal of Production Research 38: pp.2725-2742.

33. Morabito R. and Garcia V. (1997), The cutting stock problem in a hardboard industry: a case study, Computers and Operations Research 25(6): pp.469-485.
34. Padberg M. (2000), Packing small boxes into a big box,Mathematics Methods,Operations Research. 52, pp. 1–21.
35. Pierce J. F. (1964), Some Large Scale Production Scheduling Problems in the Paper lndustry, Prentice-Hall. Englewood Cliffs, New Jersey.
36. Puchinger J., Raid G.R.and Koller G. (2004), Solving a real-world glass cutting problem, In Proceedings of the 4th International Conference on Combinatorial Optimization, Coimbra, Portugal, Springer-Verlag.162-173.
37. Reinaldo M. and Luciano B. (2007), Optimizing the cutting of wood fibre plates in the hardboard industry, European Journal of Operational Research 183: pp.1405-1420.
38. Sarker, B. R. (1988), An Optimum Solution for One-dimensional Slitting Problems: A Dynamic Programming Approach, Journal of Operational Research Society, 39(8), pp. 749-755.
39. Seth A., Prasad V. R. and Ramamurthy K. G. (1986), A Heuristic Approach to One-dimensional Cutting Stock Problem., Operations Research, 23 (4), pp. 235-243.
40. Vahrenkamp, R. (1996), Random Search in the One-dimensional Cutting Stock Problem.,European Journal of Operational Research, 95 (1), pp 191-200.
41. Winston, A. E., (1994). Introduction to Mathematical Programming: (Application and Algorithms) (3rd), Duxbury Press ,California
42. Yaodong, C. and Yiping, L. (2009), Heuristic algorithm for a cutting stock problem in the steel bridge construction, Computers & Operations Research 36, pp. 612 – 622.

APPENDIX A: OUTPUT OF SAMPLE DATA BY THE APPLICATION
Order #1: Order details and output from the cutting stock application

Order length and Demand

Cut	Qty
6"	1000
4"	500
3"	119
2"	203

Strip Board Length and Cost

Strip	Cost
20"	13.5
19"	13
17"	11

Initial Solution Cutting Patterns(Fractional)

Cut	Pat1	Pat2	Pat3	Pat4
6"	3	0	0	0
4"	0	5	0	0
3"	0	0	6	0
2"	0	0	0	10
Strip	20"	20"	20"	20"
Qty	333.3	100	19.83	20.3

COST OF USED STOCK = 6391.8

Basis Inverse

0.3333333	0	0	0
0	0.2	0	0
0	0	0.1666667	0
0	0	0	0.1

Iteration #1

1. Calculate Dual variables (incremental costs)
   For length 6" Dual cost = 4.5
   For length 4" Dual cost = 2.7
   For length 3" Dual cost = 2.25
   For length 2" Dual cost = 1.35
2. Generate Entering Column by Knapsack

Cut	Pat
6"	0
4"	0
3"	5
2"	1

   Selected source length = 17
   Reduced cost = 1.6
3. Update New Column
   0
   0
   0.8333334
   0.1
4. Choose pattern to drop
   Pattern 3 is leaving, at min ratio 23.8
5. Update Basis Inverse

0.333	0	0	0
0	0.2	0	0
0	0	0.2	-0.02
0	0	0	0.1

6. Update Solution
   index : basic variable solution
   1 : 333.3333
   2 : 100
   3 : 23.8

4 : 17.92
The new cost is = 6353.72

Iteration #2

1. Calculate Dual variables (incremental costs)
   For length 6" Dual cost = 4.5
   For length 4" Dual cost = 2.7
   For length 3" Dual cost = 1.93
   For length 2" Dual cost = 1.35

2. Generate Entering Column by Knapsack

Cut	Pat
6"	3
4"	0
3"	0
2"	1

   Selected source length = 20 Reduced cost = 1.35

3. Update New Column
   1
   0
   0
   0.1

4. Choose pattern to drop
   Pattern 4 is leaving, at min ratio 179.2

5. Update Basis Inverse

0.333	0	0	0
0	0.2	0	0
0.2	0	0.2	-0.2
-1	0	0	1

6. Update Solution
   index : basic variable solution
   1 : 154.1333
   2 : 100
   3 : 23.8
   4 : 179.2
   The new cost is = 6111.8

Iteration #3

1. Calculate Dual variables (incremental costs)
   For length 6" Dual cost = 4.5
   For length 4" Dual cost = 2.7
   For length 3" Dual cost = 2.2
   For length 2" Dual cost = 0

2. Generate Entering Column by Knapsack

Cut	Pat
6"	1
4"	2
3"	1
2"	0

Selected source length = 17
Reduced cost = 1.099999

3. Update New Column

0.5333333
0.4
0.2
-0.2

4. Choose pattern to drop

Pattern 3 is leaving, at min ratio 119

5. Update Basis Inverse

0.333	0	0	0
0	0.2	0	0
-0.33	-0.4	1	0
-1	0	0	1

6. Update Solution

index : basic variable solution
1 : 90.66667
2 : 52.4
3 : 119
4 : 203
The new cost is = 5980.9

Iteration #4

1. Calculate Dual variables (incremental costs)

For length 6" Dual cost = 4.5
For length 4" Dual cost = 2.7
For length 3" Dual cost = 1.099999
For length 2" Dual cost = 0

2. Generate Entering Column by Knapsack

Cut	Pat
6"	2
4"	2
3"	0
2"	0

Selected source length = 20

Reduced cost = 0.8999996

3. Update New Column
   0.6666667
   0.4
   0
   0

4. Choose pattern to drop
   Pattern 2 is leaving, at min ratio 131

5. Update Basis Inverse
   0.333 0 0 0
   -0.33 0.5 0 0
   0.333 -1 1 0
   -1 0 0 1
6. Update Solution
   index : basic variable solution
   1 : 3.333346
   2 : 131
   3 : 119
   4 : 203
   The new cost is = 5863

Iteration #5

1. Calculate Dual variables (incremental costs)
   For length 6" Dual cost = 4.5
   For length 4" Dual cost = 2.25
   For length 3" Dual cost = 1.999999
   For length 2" Dual cost = 0

2. Generate Entering Column by Knapsack
   Cut Pat
   6" 3
   4" 0
   3" 0
   2" 0
   Selected source length = 19
   Reduced cost = 0.5
3. Update New Column
   1
   0
   0
   0

4. Choose pattern to drop
   Pattern 1 is leaving, at min ratio 3.333346
5. Update Basis Inverse

0.333	0	0	0
-0.33	0.5	0	0
0.333	-1	1	0
-1	0	0	1

6. Update Solution
   index : basic variable solution
   1 : 3.333346
   2 : 131
   3 : 119
   4 : 203
   The new cost is = 5861.333

Iteration #6

1. Calculate Dual variables (incremental costs)
   For length 6" Dual cost = 4.333333
   For length 4" Dual cost = 2.416667
   For length 3" Dual cost = 1.833333
   For length 2" Dual cost = 0.5

2. Generate Entering Column by Knapsack

Cut	Pat
6"	2
4"	1
3"	0
2"	0

   Selected source length = 17
   Reduced cost = 0.08333397

3. Update New Column
   0.3333333
   0.5
   0
   0

4. Choose pattern to drop
   Pattern 1 is leaving, at min ratio 10.00004

5. Update Basis Inverse

1	-0.5	0	0
-1	1	0	0

1 -1.5 1 0
-3 1.5 0 1

6. Update Solution

index : basic variable solution
1 : 10.00004
2 : 126
3 : 119
4 : 203
The new cost is = 5860.5

Iteration #7

1. Calculate Dual variables (incremental costs)

For length 6" Dual cost = 4.25
For length 4" Dual cost = 2.5
For length 3" Dual cost = 1.75
For length 2" Dual cost = 0.75

2. Generate Entering Column by Knapsack

Cut Pat
6" 0
4" 0
3" 0
2" 0
Selected source length = 17 Reduced cost = 0

Optimal Fractional Solution

Cut	Pat1	Pat2	Pat3	Pat4
6"	2	2	1	3
4"	1	2	2	0
3"	0	0	1	0
2"	0	0	0	1
Strip	17"	20"	17"	20"
Qty Used	10.00	126	119	203

Stock cost = 5860.5 Number of Iterations = 7

Optimal Integer Solutuion

Cut	Pat1	Pat2	Pat3	Pat4
6"	2	2	1	3
4"	1	2	2	0

3"	0	0	1	0
2"	0	0	0	1
Strip	17"	20"	17"	20"
Qty Used	10	126	119	203

Pattern #1 Stock length used: 17 Quantity: 10
2 of 6-inch Cut-piece was cut from each 17-inch stock piece
1 of 4-inch Cut-piece was cut from each 17-inch stock piece

Pattern #2 Stock length used: 20 Quantity: 126
2 of 6-inch Cut-piece was cut from each 20-inch stock piece
2 of 4-inch Cut-piece was cut from each 20-inch stock piece

Pattern #3 Stock length used: 17 Quantity: 119
1 of 6-inch Cut-piece was cut from each 17-inch stock piece
2 of 4-inch Cut-piece was cut from each 17-inch stock piece
1 of 3-inch Cut-piece was cut from each 17-inch stock piece

Pattern #4 Stock length used: 20 Quantity: 203
3 of 6-inch Cut-piece was cut from each 20-inch stock piece
1 of 2-inch Cut-piece was cut from each 20-inch stock piece

Optimal Stock Used = 458; Optimal Stock cost = 5860.5

Order #2: Order details and output from the cutting stock application

Order length and Demand

Cut	Qty
6"	540
4"	850
3"	40
2"	85

Strip Board Length and Cost

Strip	Cost

20"	13.5
19"	13
17"	11

Initial Solution Cutting Patterns (Fractional)

Cut	Pat1	Pat2	Pat3	Pat4
6"	3	0	0	0
4"	0	5	0	0
3"	0	0	6	0
2"	0	0	0	10
Strip	20"	20"	20"	20"
Qty	180	170	6.666667	8.5

COST OF USED STOCK = 4929.75

Basis Inverse

0.3333333	0	0	0
0	0.2	0	0
0	0	0.1666667	0
0	0	0	0.1

Iteration #1

1. Calculate Dual variables (incremental costs)
   - For length 6" Dual cost = 4.5
   - For length 4" Dual cost = 2.7
   - For length 3" Dual cost = 2.25
   - For length 2" Dual cost = 1.35

2. Generate Entering Column by Knapsack

Cut	Pat
6"	0
4"	0
3"	5

2" 1
Selected source length = 17
Reduced cost = 1.6

3. Update New Column
   0
   0
   0.8333334
   0.1

4. Choose pattern to drop
   Pattern 3 is leaving, at min ratio 8

5. Update Basis Inverse

0.3333	0	0	0
0	0.2	0	0
0	0	0.2	-0.02
0	0	0	0.1

6. Update Solution
   index : basic variable solution
   1 : 180
   2 : 170
   3 : 8
   4 : 7.7
   The new cost is = 4916.95

Iteration #2

1. Calculate Dual variables (incremental costs)
   For length 6" Dual cost = 4.5
   For length 4" Dual cost = 2.7
   For length 3" Dual cost = 1.93
   For length 2" Dual cost = 1.35

2. Generate Entering Column by Knapsack
   Cut Pat
   6" 3
   4" 0
   3" 0
   2" 1
   Selected source length = 20
   Reduced cost = 1.35
3. Update New Column
   1
   0
   0
   0.1

4. Choose pattern to drop
   Pattern 4 is leaving, at min ratio 77

5. Update Basis Inverse

0.3333	0	0	0
0	0.2	0	0
0.2	0	0.2	-0.2
-1	0	0	1

6. Update Solution
   index : basic variable solution
   1 : 103
   2 : 170
   3 : 8
   4 : 77
   The new cost is = 4813

Iteration #3

1. Calculate Dual variables (incremental costs)
   For length 6" Dual cost = 4.5
   For length 4" Dual cost = 2.7
   For length 3" Dual cost = 2.2
   For length 2" Dual cost = 0

2. Generate Entering Column by Knapsack

Cut	Pat
6"	1
4"	2
3"	1
2"	0

   Selected source length = 17
   Reduced cost = 1.099999

3. Update New Column
   0.5333333
   0.4
   0.2
   -0.2
4. Choose pattern to drop
   Pattern 3 is leaving, at min ratio 40
5. Update Basis Inverse

0.3333	0	0	0
0	0.2	0	0
-0.333	-0.4	1	0
-1	0	0	1

6. Update Solution

   index : basic variable solution

   1 : 81.66666

   2 : 154

   3 : 40

   4 : 85

   The new cost is = 4769

Iteration #4

1. Calculate Dual variables (incremental costs)

   For length 6" Dual cost = 4.5

   For length 4" Dual cost = 2.7

   For length 3" Dual cost = 1.099999

   For length 2" Dual cost = 0

2. Generate Entering Column by Knapsack

Cut	Pat
6"	2
4"	2
3"	0
2"	0

   Selected source length = 20

   Reduced cost = 0.8999996

3. Update New Column

   0.6666667

   0.4

   0

   0

4. Choose pattern to drop

   Pattern 1 is leaving, at min ratio 122.5

5. Update Basis Inverse

0.5	-0.2	0	0
0	0.2	0	0
-0.5	-0.2	1	0
-1.5	0.6	0	1

6. Update Solution

   index : basic variable solution

   1 : 122.5

   2 : 105

   3 : 40

4 : 85
The new cost is = 4658.75

Iteration #5

1. Calculate Dual variables (incremental costs)
   For length 6" Dual cost = 4.05
   For length 4" Dual cost = 2.7
   For length 3" Dual cost = 1.549999
   For length 2" Dual cost = 1.349999

2. Generate Entering Column by Knapsack
   Cut Pat
   6" 0
   4" 0
   3" 0
   2" 0
   Selected source length = 20
   Reduced cost = 0

Optimal Fractional Solution

Cut	Pat1	Pat2	Pat3	Pat4
6"	2	0	1	3
4"	2	5	2	0
3"	0	0	1	0
2"	0	0	0	1
Strip	20"	20"	17"	20"
Qty Used	122.5	105	40	85

Stock cost = 4658.75 Number of Iterations = 5

Optimal Integer Solutuion

Cut	Pat1	Pat2	Pat3	Pat4
6"	2	0	1	3
4"	2	5	2	0
3"	0	0	1	0
2"	0	0	0	1

Strip	20"	20"	17"	20"
Qty Used	123	105	40	85

Pattern #1 Stock length used: 20 Quantity: 123
2 of 6-inch Cut-piece was cut from each 20-inch stock piece
2 of 4-inch Cut-piece was cut from each 20-inch stock piece

Pattern #2 Stock length used: 20 Quantity: 105
5 of 4-inch Cut-piece was cut from each 20-inch stock piece

Pattern #3 Stock length used: 17 Quantity: 40
1 of 6-inch Cut-piece was cut from each 17-inch stock piece
2 of 4-inch Cut-piece was cut from each 17-inch stock piece
1 of 3-inch Cut-piece was cut from each 17-inch stock piece

Pattern #4 Stock length used: 20 Quantity: 85
3 of 6-inch Cut-piece was cut from each 20-inch stock piece
1 of 2-inch Cut-piece was cut from each 20-inch stock piece

Optimal Stock Used = 353; Optimal Stock cost = 4665.5

Order #3: Order details and output from the cutting stock application

Order length and Demand

Cut	Qty
6"	1500
4"	200
2"	203

Strip Board Length and Cost

Strip	Cost
20"	13.5
19"	13
17"	11

Initial Solution Cutting Patterns(Fractional)

Cut	Pat1	Pat2	Pat3
6"	3	0	0
4"	0	5	0
2"	0	0	10
Strip	20"	20"	20"
Qty Used	500	40	20.3

COST OF USED STOCK = 7564.05

Basis Inverse

0.3333 0 0
0 0.2 0
0 0 0.1

Iteration #1

1. Calculate Dual variables (incremental costs)
   For length 6" Dual cost = 4.5
   For length 4" Dual cost = 2.7
   For length 2" Dual cost = 1.35

2. Generate Entering Column by Knapsack
   Cut Pat
   6" 3
   4" 0
   2" 1
   Selected source length = 20
   Reduced cost = 1.35
3. Update New Column
   1
   0
   0.1
4. Choose pattern to drop
   Pattern 3 is leaving, at min ratio 203

5. Update Basis Inverse
   0.3333 0 0
   0 0.2 0
   -1 0 1
6. Update Solution
   index : basic variable solution
   1 : 297
   2 : 40
   3 : 203
   The new cost is = 7290

Iteration #2

1. Calculate Dual variables (incremental costs)
   For length 6" Dual cost = 4.5
   For length 4" Dual cost = 2.7
   For length 2" Dual cost = 0

2. Generate Entering Column by Knapsack
   Cut Pat
   6" 2
   4" 2
   2" 0
   Selected source length = 20
   Reduced cost = 0.8999996

3. Update New Column
   0.6666667

0.4
0

4. Choose pattern to drop
   Pattern 2 is leaving, at min ratio 100

5. Update Basis Inverse
   0.3333 0 0
   -0.333 0.5 0
   -1 0 1

6. Update Solution
   index : basic variable solution
   1 : 230.3333
   2 : 100
   3 : 203
   The new cost is = 7200

Iteration #3

1. Calculate Dual variables (incremental costs)
   For length 6" Dual cost = 4.5
   For length 4" Dual cost = 2.25
   For length 2" Dual cost = 0

2. Generate Entering Column by Knapsack
   Cut Pat
   6" 3
   4" 0
   2" 0
   Selected source length = 19
   Reduced cost = 0.5

3. Update New Column
   1
   0
   0

4. Choose pattern to drop
   Pattern 1 is leaving, at min ratio 230.3333

5. Update Basis Inverse
   0.3333 0 0
   -0.333 0.5 0

-1 0 1

6. Update Solution

index : basic variable solution
1 : 230.3333
2 : 100
3 : 203
The new cost is = 7084.833

Iteration #4

1. Calculate Dual variables (incremental costs)

For length 6" Dual cost = 4.333333
For length 4" Dual cost = 2.416667
For length 2" Dual cost = 0.5

2. Generate Entering Column by Knapsack

Cut Pat
6" 2
4" 1
2" 0
Selected source length = 17
Reduced cost = 0.08333397

3. Update New Column

0.3333333
0.5
0

4. Choose pattern to drop

Pattern 2 is leaving, at min ratio 200

5. Update Basis Inverse

0.3333 0 0
-0.666 1 0
-1 0 1

6. Update Solution

index : basic variable solution
1 : 163.6667
2 : 200
3 : 203
The new cost is = 7068.166

Iteration #5

1. Calculate Dual variables (incremental costs)

For length 6" Dual cost = 4.333333

For length 4" Dual cost = 2.333333
For length 2" Dual cost = 0.5

2. Generate Entering Column by Knapsack

Cut Pat
6" 0
4" 0
2" 0
Selected source length = 17
Reduced cost = 0

Optimal Fractional Solution

Cut	Pat1	Pat2	Pat3
6"	3	2	3
4"	0	1	0
2"	0	0	1
Strip	19"	17"	20"
Qty Used	163.6667	200	203

Stock cost = 7068.167 Number of Iterations = 5

Optimal Integer Solutuion

Cut	Pat1	Pat2	Pat3
6"	3	2	3
4"	0	1	0
2"	0	0	1
Strip	19"	17"	20"
Qty Used	164	200	203

Pattern #1 Stock length used: 19 Quantity: 164
3 of 6-inch Cut-piece was cut from each 19-inch stock piece

Pattern #2 Stock length used: 17 Quantity: 200
2 of 6-inch Cut-piece was cut from each 17-inch stock piece
1 of 4-inch Cut-piece was cut from each 17-inch stock piece

Pattern #3 Stock length used: 20 Quantity: 203
3 of 6-inch Cut-piece was cut from each 20-inch stock piece
1 of 2-inch Cut-piece was cut from each 20-inch stock piece

Optimal Stock Used = 567; Optimal Stock cost = 7072.5

Order #4: Order details and output from the cutting stock application

Order length and Demand

Cut	Qty
6"	200
4"	150
3"	100
2"	120

Strip Board Length and Cost

Strip	Cost
20"	13.5
19"	13
17"	11

Initial Solution Cutting Patterns(Fractional)

Cut	Pat1	Pat2	Pat3	Pat4
6"	3	0	0	0
4"	0	5	0	0
3"	0	0	6	0
2"	0	0	0	10
Strip	20"	20"	20"	20"
Qty Used	66.66667	30	16.66667	12

COST OF USED STOCK = 1692

Basis Inverse

0.3333	0	0	0
0	0.2	0	0
0	0	0.1666	0

0 0 0 0.1

Iteration #1

1. Calculate Dual variables (incremental costs)
   For length 6" Dual cost = 4.5
   For length 4" Dual cost = 2.7
   For length 3" Dual cost = 2.25
   For length 2" Dual cost = 1.35

2. Generate Entering Column by Knapsack
   Cut Pat
   6" 0
   4" 0
   3" 5
   2" 1
   Selected source length = 17
   Reduced cost = 1.6

3. Update New Column
   0
   0
   0.8333334
   0.1
4. Choose pattern to drop
   Pattern 3 is leaving, at min ratio 20

5. Update Basis Inverse
   0.3333 0 0 0
   0 0.2 0 0
   0 0 0.2 -0.02
   0 0 0 0.1

6. Update Solution
   index : basic variable solution
   1 : 66.66667
   2 : 30
   3 : 20
   4 : 10
   The new cost is = 1660

Iteration #2

1. Calculate Dual variables (incremental costs)

For length 6" Dual cost = 4.5
For length 4" Dual cost = 2.7
For length 3" Dual cost = 1.93
For length 2" Dual cost = 1.35

2. Generate Entering Column by Knapsack

Cut	Pat
6"	3
4"	0
3"	0
2"	1

Selected source length = 20
Reduced cost = 1.35

3. Update New Column

1
0
0
0.1

4. Choose pattern to drop

Pattern 1 is leaving, at min ratio 66.66667

5. Update Basis Inverse

0.3333	0	0	-0.03333334
0	0.2	0	0
0	0	0.2	-0.02
0	0	0	0.1

6. Update Solution

index : basic variable solution
1 : 66.66667
2 : 30
3 : 20
4 : 3.333333
The new cost is = 1570

Iteration #3

1. Calculate Dual variables (incremental costs)

For length 6" Dual cost = 4.05
For length 4" Dual cost = 2.7
For length 3" Dual cost = 1.93
For length 2" Dual cost = 1.35

2. Generate Entering Column by Knapsack

Cut	Pat
6"	0
4"	1

3" 1
2" 5
Selected source length = 17
Reduced cost = 0.3800001

3. Update New Column
   0
   0.2
   0.2
   0.48
4. Choose pattern to drop
   Pattern 4 is leaving, at min ratio 6.944443
5. Update Basis Inverse
   0.3333 0.0139 0.0138 -0.06944445
   0 0.2 0 0
   0 0.0083 0.2083 -0.04166666
   0 -0.041 -0.041 0.2083333

6. Update Solution
   index : basic variable solution
   1 : 66.66667
   2 : 28.61111
   3 : 18.61111
   4 : 6.944443
   The new cost is = 1567.361

Iteration #4

1. Calculate Dual variables (incremental costs)
   For length 6" Dual cost = 4.076389
   For length 4" Dual cost = 2.7
   For length 3" Dual cost = 1.945833
   For length 2" Dual cost = 1.270833

2. Generate Entering Column by Knapsack
   Cut Pat
   6" 1
   4" 2
   3" 1
   2" 0
   Selected source length = 17
   Reduced cost = 0.4222221

3. Update New Column
   0.3333333
   0.4222222

0.2222222
-0.1111111

4. Choose pattern to drop
   Pattern 2 is leaving, at min ratio 67.76315

5. Update Basis Inverse

0.3223684	0.03289474	0.006578947	-0.06578948
-0.1578947	0.4736842	-0.1052632	0.05263158
-0.006578947	0.01973684	0.2039474	-0.03947368
0.03289474	-0.09868421	-0.01973684	0.1973684

6. Update Solution
   index : basic variable solution
   1 : 44.07895
   2 : 67.76315
   3 : 3.552633
   4 : 14.47368
   The new cost is = 1538.75

Iteration #5

1. Calculate Dual variables (incremental costs)
   For length 6" Dual cost = 4.0625
   For length 4" Dual cost = 2.5
   For length 3" Dual cost = 1.9375
   For length 2" Dual cost = 1.3125

2. Generate Entering Column by Knapsack

Cut	Pat
6"	2
4"	0
3"	1
2"	1

   Selected source length = 17
   Reduced cost = 0.375001

3. Update New Column
   0.6710527
   -0.0131579
   0.1973684
   0.02631578
4. Choose pattern to drop
   Pattern 3 is leaving, at min ratio 18.00001

5. Update Basis Inverse

0.3 0.0333 0.0333 -0.06666667
0.2 0.4666 -0.533 0.06666666
-0.70 0.0333 1.0333 -0.06666666
0.1 -0.1 -0.1 0.2

6. Update Solution

index : basic variable solution
1 : 32
2 : 67.99999
3 : 18.00001
4 : 14
The new cost is = 1532

Iteration #6

1. Calculate Dual variables (incremental costs)

For length 6" Dual cost = 4.05
For length 4" Dual cost = 2.7
For length 3" Dual cost = 1.549998
For length 2" Dual cost = 1.35

2. Generate Entering Column by Knapsack

Cut Pat
6" 0
4" 0
3" 0
2" 10
Selected source length = 20
Reduced cost = 1.907349E-06

Optimal Fractional Solution

Cut	Pat1	Pat2	Pat3	Pat4
6"	3	1	2	0
4"	0	2	0	1
3"	0	1	1	1
2"	1	0	1	5
Strip	20"	17"	17"	17"
Qty Used	32	67.99999	18.00001	14

Stock cost = 1532 Number of Iterations = 6

Optimal Integer Solutuion

Cut	Pat1	Pat2	Pat3	Pat4
6"	3	1	2	0
4"	0	2	0	1
3"	0	1	1	1
2"	1	0	1	5
Strip	20"	17"	17"	17"
Qty Used	32	68	18	14

Pattern #1 Stock length used: 20 Quantity: 32
3 of 6-inch Cut-piece was cut from each 20-inch stock piece
1 of 2-inch Cut-piece was cut from each 20-inch stock piece

Pattern #2 Stock length used: 17 Quantity: 68
1 of 6-inch Cut-piece was cut from each 17-inch stock piece
2 of 4-inch Cut-piece was cut from each 17-inch stock piece
1 of 3-inch Cut-piece was cut from each 17-inch stock piece

Pattern #3 Stock length used: 17 Quantity: 18
2 of 6-inch Cut-piece was cut from each 17-inch stock piece
1 of 3-inch Cut-piece was cut from each 17-inch stock piece
1 of 2-inch Cut-piece was cut from each 17-inch stock piece

Pattern #4 Stock length used: 17 Quantity: 14
1 of 4-inch Cut-piece was cut from each 17-inch stock piece
1 of 3-inch Cut-piece was cut from each 17-inch stock piece
5 of 2-inch Cut-piece was cut from each 17-inch stock piece

Optimal Stock Used = 132; Optimal Stock cost = 1532

Order #5: Order details and output from the cutting stock application

Order length and Demand

Cut	Qty
6"	220
4"	115
3"	120
2"	40

Strip Board Length and Cost

Strip	Cost
20"	13.5
19"	13
17"	11

Initial Solution Cutting Patterns(Fractional)

Cut	Pat1	Pat2	Pat3	Pat4
6"	3	0	0	0
4"	0	5	0	0
3"	0	0	6	0
2"	0	0	0	10
Strip	20"	20"	20"	20"
Qty	73.33334	23	20	4

COST OF USED STOCK = 1624.5

Basis Inverse

0.3333	0	0	0
0	0.2	0	0
0	0	0.1666	0

0 0 0 0.1

Iteration #1

1. Calculate Dual variables (incremental costs)
   For length 6" Dual cost = 4.5
   For length 4" Dual cost = 2.7
   For length 3" Dual cost = 2.25
   For length 2" Dual cost = 1.35

2. Generate Entering Column by Knapsack
   Cut Pat
   6" 0
   4" 0
   3" 5
   2" 1
   Selected source length = 17
   Reduced cost = 1.6

3. Update New Column
   0
   0
   0.8333334
   0.1
4. Choose pattern to drop
   Pattern 3 is leaving, at min ratio 24

5. Update Basis Inverse
   0.3333 0 0 0
   0 0.2 0 0
   0 0 0.2 -0.02
   0 0 0 0.1
6. Update Solution
   index : basic variable solution
   1 : 73.33334
   2 : 23
   3 : 24
   4 : 1.6
   The new cost is = 1586.1

Iteration #2

1. Calculate Dual variables (incremental costs)
   For length 6" Dual cost = 4.5

For length 4" Dual cost = 2.7
For length 3" Dual cost = 1.93
For length 2" Dual cost = 1.35

2. Generate Entering Column by Knapsack

Cut	Pat
6"	3
4"	0
3"	0
2"	1

Selected source length = 20
Reduced cost = 1.35

3. Update New Column

1
0
0
0.1

4. Choose pattern to drop

Pattern 4 is leaving, at min ratio 16

5. Update Basis Inverse

0.3333	0	0	0
0	0.2	0	0
0.2	0	0.2	-0.2
-1	0	0	1

6. Update Solution

index : basic variable solution
1 : 57.33334
2 : 23
3 : 24
4 : 16
The new cost is = 1564.5

Iteration #3

1. Calculate Dual variables (incremental costs)

For length 6" Dual cost = 4.5
For length 4" Dual cost = 2.7
For length 3" Dual cost = 2.2
For length 2" Dual cost = 0

2. Generate Entering Column by Knapsack

Cut	Pat
6"	1

4" 2
3" 1
2" 0
Selected source length = 17
Reduced cost = 1.099999

3. Update New Column
   0.5333333
   0.4
   0.2
   -0.2
4. Choose pattern to drop
   Pattern 2 is leaving, at min ratio 57.5
5. Update Basis Inverse

0.3333	0	0	0
-0.266	0.5	-0.099	0.099
0.2	0	0.2	-0.2
-1	0	0	1

6. Update Solution
   index : basic variable solution
   1 : 26.66667
   2 : 57.5
   3 : 12.5
   4 : 27.5
   The new cost is = 1501.25

Iteration #4

1. Calculate Dual variables (incremental costs)
   For length 6" Dual cost = 4.5
   For length 4" Dual cost = 2.15
   For length 3" Dual cost = 2.2
   For length 2" Dual cost = 0

2. Generate Entering Column by Knapsack
   Cut Pat
   6" 3
   4" 0
   3" 0
   2" 0
   Selected source length = 19
   Reduced cost = 0.5

3. Update New Column
   1
   0

0
0

4. Choose pattern to drop
   Pattern 1 is leaving, at min ratio 26.66667

5. Update Basis Inverse
   0.3333 0 0 0
   -0.266 0.5 -0.099 0.099
   0.2 0 0.2 -0.2
   -1 0 0 1

6. Update Solution
   index : basic variable solution
   1 : 26.66667
   2 : 57.5
   3 : 12.5
   4 : 27.5
   The new cost is = 1487.917

Iteration #5

1. Calculate Dual variables (incremental costs)
   For length 6" Dual cost = 4.333333
   For length 4" Dual cost = 2.283334
   For length 3" Dual cost = 2.1
   For length 2" Dual cost = 0.5

2. Generate Entering Column by Knapsack
   Cut Pat
   6" 2
   4" 0
   3" 1
   2" 1
   Selected source length = 17
   Reduced cost = 0.2666664

3. Update New Column
   -0.1333333
   0
   0.2
   0.8
4. Choose pattern to drop
   Pattern 4 is leaving, at min ratio 34.375

5. Update Basis Inverse

0.3333	0	0	0
-0.25	0.5	-0.125	0.125
0.1666	0	0.25	-0.25
-0.8333	0	-0.25	1.25

6. Update Solution

   index : basic variable solution
   1 : 31.25001
   2 : 57.5
   3 : 5.625
   4 : 34.375
   The new cost is = 1478.75

Iteration #6

1. Calculate Dual variables (incremental costs)

   For length 6" Dual cost = 4.333333
   For length 4" Dual cost = 2.25
   For length 3" Dual cost = 2.166667
   For length 2" Dual cost = 0.166666

2. Generate Entering Column by Knapsack

Cut	Pat
6"	0
4"	1
3"	5
2"	0

   Selected source length = 19
   Reduced cost = 0.08333397

3. Update New Column

   0.5833333
   0.5
   1.125
   -1.125

4. Choose pattern to drop

   Pattern 3 is leaving, at min ratio 5

5. Update Basis Inverse

0.3333333	0	0	0
-0.1851852	0.5555556	-0.1111111	0
0.03703702	-0.1111111	0.2222222	0
-0.7037038	0.1111111	-0.2222222	1

6. Update Solution

   index : basic variable solution

1 : 28.33334
2 : 55
3 : 5
4 : 40
The new cost is = 1478.333

Iteration #7

1. Calculate Dual variables (incremental costs)
   For length 6" Dual cost = 4.333333
   For length 4" Dual cost = 2.25926
   For length 3" Dual cost = 2.148148
   For length 2" Dual cost = 0.1851845
2. Generate Entering Column by Knapsack

Cut	Pat
6"	2
4"	1
3"	1
2"	0

   Selected source length = 19
   Reduced cost = 0.0740757

3. Update New Column
   0.5185186
   0.4444445
   0.1111111
   0
4. Choose pattern to drop
   Pattern 3 is leaving, at min ratio 45

5. Update Basis Inverse

0.3333333	0	0	0
0.3333334	1	-1	0
-1	-1	2	0
0.3333333	1	-2	1

6. Update Solution
   index : basic variable solution
   1 : 5.000004
   2 : 35
   3 : 45
   4 : 40
   The new cost is = 1475

Iteration #8

1. Calculate Dual variables (incremental costs)
   For length 6" Dual cost = 4.333333

For length 4" Dual cost = 2.333336
For length 3" Dual cost = 1.999996
For length 2" Dual cost = 0.3333349

2. Generate Entering Column by Knapsack

Cut	Pat
6"	2
4"	1
3"	0
2"	0

Selected source length = 17
Reduced cost = 2.861023E-06

Optimal Fractional Solution

Cut	Pat1	Pat2	Pat3	Pat4
6"	3	1	2	2
4"	0	2	1	0
3"	0	1	1	1
2"	0	0	0	1
Strip	19"	17"	19"	17"
Qty Used	5.000004	35	45	40

Stock cost = 1475 Number of Iterations = 8

Optimal Integer Solutuion

Cut	Pat1	Pat2	Pat3	Pat4
6"	3	1	2	2
4"	0	2	1	0
3"	0	1	1	1
2"	0	0	0	1
Strip	19"	17"	19"	17"
Qty Used	5	35	45	40

Pattern #1 Stock length used: 19 Quantity: 5
3 of 6-inch Cut-piece was cut from each 19-inch stock piece

Pattern #2 Stock length used: 17 Quantity: 35
1 of 6-inch Cut-piece was cut from each 17-inch stock piece
2 of 4-inch Cut-piece was cut from each 17-inch stock piece
1 of 3-inch Cut-piece was cut from each 17-inch stock piece

Pattern #3 Stock length used: 19 Quantity: 45
2 of 6-inch Cut-piece was cut from each 19-inch stock piece
1 of 4-inch Cut-piece was cut from each 19-inch stock piece
1 of 3-inch Cut-piece was cut from each 19-inch stock piece

Pattern #4 Stock length used: 17 Quantity: 40
2 of 6-inch Cut-piece was cut from each 17-inch stock piece
1 of 3-inch Cut-piece was cut from each 17-inch stock piece
1 of 2-inch Cut-piece was cut from each 17-inch stock piece

Optimal Stock Used = 125; Optimal Stock cost = 1475

APPENDIX B - VISUAL BASIC.NET CODES FOR THE CUTTING STOCK

```
‘Declarations

Public Class ColumnGeneration

    Implements ICloneable

    Private NPart, NSource As Integer  '  Nbr of required parts and supply parts

    Public Verbose, Fractional As Boolean

    Public StockLengths() As Integer

    Public StockCost() As Single

    Public StockUsed() As Integer '  Stock lengths, Stock costs, Stock Used

    Public StockOrigLoc() As Integer '  Used to report usage back in the right row

    Public Partlengths() As Single

    Public PartQty() As Single '   required part lengths and quanties

    Public totparts() As Integer '  for each length, sum of parts cut from all patterns

    Public StockusedByPattern() As Single '   required nbr of pieces for each pattern

    Public IStockUsedByPattern() As Integer '  integer version

    Public A(,) As Integer ' array of  columns (patterns) and rows (part lengths)

    Public C1() As Single

    Public result As Boolean

    Public tag As Integer

    Public msg As String = ""   'Contains output info

    '--------------------  INPUT DATA  ------------------------

    Private SL(), SC() As Single    'lengths  and costs of stock piece types

    '--------------------  SIMPLEX DATA  ------------------------*

    Private BI(,) As Single  ' : ARRAY of array of extended;  Inverse of B

    Private BBAR() As Single    ';  required amount of ith length
```

```
Private B() As Single '  Right hand side
Private AR() As Single '   updated entering column
Private ARINV() As Single '  Inverse of AR
Private PI() As Single '  Simplex multiplier (Dual cost)
Private ZB As Single    '  Total cost
Private RMAX As Single '  minimum ratio of ratio test
Private CMIN1 As Single ' minimum value of j-th reduced cost
Private NR, NROW As Integer '   number of constraints
'--------------------  ENTCOL DATA  --------------------------
Private SCOST As Single
Private SSOURCE As Integer
'--------------------  KNAPSACK DATA  ----------------------
Private KA() As Single '
Private STATE(), VARS(), XS() As Integer
Private NODE(,), D(,) As Integer
Private Demand(,), Inv(,) As Single
Private RESOURCE As Integer
Private NCOL, NSTAGE, OPTN As Integer
Private C() As Single '
Private COST(,) As Single
Private F(,) As Single '
Private OPTD As Integer
Private OPTF As Single
```

```
'--------------------  control variables  --------------------
Dim SW As Char
Dim fp As String
Dim IERROR As Integer
Dim ERR_MSG As String
Public SLENGTH() As Single    ' Stock lengths selected for each pattern
Public Tolerance As Single = 0.00001
Public IT As Integer

Public Sub GetData(ByVal M As Integer, ByVal N As Integer, ByVal stL() As  _ Single,ByVal stC() As
    Single, ByVal iL() As Single, ByVal iQ() As Single)
  Dim i, j As Integer
  NSource = N
  NPart = M
  Init(M, N)
  Verbose = True
  Fractional = True
  For i = 1 To NSource
    StockLengths(i) = stL(i - 1)
    StockCost(i) = stC(i - 1)
    StockOrigLoc(i) = i
    StockUsed(i) = 0
  Next
  For i = 1 To NPart
    For j = 1 To NPart : A(i, j) = 0 : Next j
```

```
    Next i
    For i = 1 To NPart
        Partlengths(i) = iL(i - 1)
        PartQty(i) = iQ(i - 1)
    Next
End Sub

Public Sub Solve()
    Dim i As Integer
    For i = 1 To NSource
        SL(i) = StockLengths(i)
        SC(i) = StockCost(i)
        StockOrigLoc(i) = StockOrigLoc(i)
        StockUsed(i) = StockUsed(i)
    Next i
    Verbose = True
    Fractional = True
    For i = 1 To NPart
        Partlengths(i) = Partlengths(i)
        PartQty(i) = PartQty(i)
   Next
    ReDim NODE(NPart + 1, 0)
    ReDim Demand(NPart + 1, 0) ' max nbr of shortest piece from longest stock
    ReDim D(NPart + 1, 0)
```

```
    ReDim Inv(NPart + 1, 0)
    ReDim C(NPart + 1)
    ReDim COST(NPart + 1, 0)
    ReDim F(NPart + 1, 0)
    NROW = NPart
    IT = 1
    InitialSolution()
    DualVariables()
    EnteringColumn()
    While (CMIN1 <= -Tolerance)
       UpdateEnteringColumn()
       LeavingColumn()
       UpdateBInverse()
       UpdateSolution()
       IT = IT + 1
       ' a column change :'Updates the patterns after a new pattern has been found
       For i = 1 To NPart : A(NR, i) = XS(i) : Next i
       ' a cost endowing
       C1(NR) = SCOST
       SLENGTH(NR) = SL(SSOURCE)
       DualVariables()     'Computes dual variables
       EnteringColumn()
    End While
    OptimalSolution()  ' display solution
End Sub
```

```
Private Sub InitialSolution()
    ' find an initial solution.
    Dim i, j As Integer
    For i = 1 To NPart
        For j = 1 To NPart
            If (i <> j) Then : A(i, j) = 0
                'Finds the initial m patterns for
                                the application
            Else : A(i, j) = Math.Truncate(SL(1) / Partlengths(i))
                                '    use first source
            End If
        Next j
    Next i

    For i = 1 To NPart
        For j = 1 To NPart
            'Creates the inverse of the initial
                                Basis B (in this case A) [ BI ]
            If (i = j) And (A(j, i) > 0) Then : BI(j, i) = 1.0 / A(j, i)
            Else : BI(j, i) = 0
            End If
        Next j
    Next i
```

```
    For i = 1 To NPart
        StockusedByPattern(i) = 0
        'Compute the total stocks used by each pattern i
                    For j = 1 To NPart : StockusedByPattern(i) = StockusedByPattern(i) + BI(j, i) *
                    PartQty(j) : Next j
    Next i
    For i = 1 To NPart
        C1(i) = SC(1)
        SLENGTH(i) = SL(1)
    Next i
    ZB = 0
    For i = 1 To NPart : ZB = ZB + C1(i) * StockusedByPattern(i) : Next i
    PrintInitialSolution()  'Prints the initial solution
    msg &= vbNewLine
    msg &= " ---  B Inverse  --- " & vbNewLine
    Me.PrintMatrix(BI)      'Prints the inverse of the B Matrix
    msg &= vbNewLine
End Sub


Private Sub DualVariables()
    'computes simplex multiplier
    Dim i, j As Integer
    For i = 1 To NPart
        PI(i) = 0
```

```
        For j = 1 To NPart : PI(i) = PI(i) + C1(j) * BI(j, i) : Next
    Next i
    If Verbose Then
        msg += " " & vbNewLine
        msg += " " & vbNewLine
        msg += "********** Iteration #" & CStr(IT) _
                        & "**********" & vbNewLine
        msg += "--- Calculate Dual variables (incremental costs) _
                        ---" & vbNewLine
        For i = 1 To NPart
            msg &= "For length  " & CStr(Partlengths(i))
            msg &= "      Dual cost = " & PI(i) & vbNewLine
        Next i
    End If
End Sub

Private Sub EnteringColumn()
    ' finds a cutting pattern to improve current
                ‘solution by dynamic program algorithm
    Dim i, j, k As Integer
    Dim tempx() As Integer
    Dim Valu, maxval As Single
    ReDim tempx(NPart + 1)
    For i = 1 To NPart : XS(i) = 0 : Next i '   Check Pi_value
                        ‘before column generation
```

```
' Check Pi_value before column generation not implimented yet.
' SLACK ENTERING
For i = 1 To NPart
   If (PI(i) < -Tolerance) Then
      XS(i) = -1
      SCOST = 0.0
      For j = 1 To NSource
         If (Partlengths(i) <= SL(j)) Then SSOURCE = j
      Next
      If Verbose Then
         For j = 1 To NPart '
            If XS(j) > 0 Then
               For k = 1 To XS(j) '
               Next k
            End If
         Next j
      End If
   End If
Next
' Find integer solution  by dynamic programming
maxval = 0.0
' for NSOURCE type of source length
For k = 1 To NSource 'DP data formation
   NCOL = NPart
   RESOURCE = Math.Truncate(SL(k))
```

```
    For i = 1 To NPart : KA(i) = Partlengths(i) : Next i
    For i = 1 To NPart : C(i) = PI(i) : Next i
    ' Call dynamic program
    DPKnapsack()
    ' Check optimal condition  Zj -Cj <= 0
    ' Select source type by maxval
    Valu = 0.0
    For i = 1 To NPart : Valu = Valu + XS(i) * PI(i) : Next i
    If ((Valu - SC(k)) > maxval) Then
       maxval = Valu - SC(k)
       SSOURCE = k
       SCOST = SC(k)
       ' Return a generated entering column
       For j = 1 To NPart : tempx(j) = XS(j) : Next j
    End If
  Next k  ' End of Selection loop
  For j = 1 To NPart : XS(j) = tempx(j) : Next j
  CMIN1 = -maxval
  PrintSolution(maxval)
End Sub
```

```
Private Sub DPKnapsack()
   '  solves a knapsack problem using dynamic programming
   Dim i, j, k, nvar, nstate, nstate1, stage, _
                  stage1, invent, tempn As Integer
   Dim temp As Single
   'Dim s As String
   ' formulation
   NSTAGE = NCOL
   STATE(1) = 1
   If RESOURCE >= UBound(NODE, 1) Then
      '   increase NState array sizes
      ReDim NODE(NPart + 1, RESOURCE + 2)
      ReDim Demand(NPart + 1, RESOURCE + 2) ' max nbr of
                        'shortest piece from longest stock )
      ReDim Inv(NPart + 1, RESOURCE + 2)
      ReDim COST(NPart + 1, RESOURCE + 2)
      ReDim F(NPart + 1, RESOURCE + 2)
      ReDim D(NPart + 1, RESOURCE + 2)
   End If
   For i = 1 To NSTAGE : VARS(i) = Math.Truncate(RESOURCE / _
                  KA(i) + 1) : Next
   For i = 2 To NCOL : STATE(i) = RESOURCE + 1 : Next
   For i = 1 To NSTAGE
      nstate = VARS(i)
      For j = 1 To nstate : NODE(i, j) = j : Next
```

```
Next i
For i = 1 To NCOL
   nstate = VARS(i)
   For j = 2 To nstate : Demand(i, j) = (j - 1) * KA(i) : Next
   Demand(i, 1) = 0
Next i
For i = 1 To NCOL
   nstate = VARS(i)
   For j = 2 To nstate : COST(i, j) = (j - 1) * C(i) : Next
   COST(i, 1) = 0
Next i
' initialization
stage = NSTAGE
nstate = STATE(NSTAGE)
nstate1 = VARS(stage)
For j = 1 To nstate
   F(stage, j) = 0
   D(stage, j) = 0
Next
For j = 1 To nstate
   For k = 1 To nstate1
      invent = RESOURCE - (j - 1)
      If (invent >= Demand(stage, k)) Then
         temp = COST(stage, k)
         If (F(stage, j) < temp) Then
```

```
            F(stage, j) = temp
            D(stage, j) = k
            Inv(stage, j) = invent - Demand(stage, k)
          End If
        End If
    Next
Next

' DP loop
For i = 2 To NSTAGE
    stage = NSTAGE - i + 1
    stage1 = stage + 1  ' backward
    nvar = VARS(stage)
    nstate = STATE(stage)
    For j = 1 To nstate
        F(stage, j) = 0
        NODE(stage, j) = 0
        D(stage, j) = 0
    Next j
    For j = 1 To nstate
        invent = 0
        For k = 1 To nvar
            invent = RESOURCE - (j - 1)
            If (invent >= Demand(stage, k)) Then
                invent = invent - Demand(stage, k)
```

```
            tempn = Math.Round(RESOURCE - invent + 1)   ' GDD
            temp = COST(stage, k) + F(stage1, tempn)
            If (F(stage, j) < temp) Then
               F(stage, j) = temp
               D(stage, j) = k
               NODE(stage, j) = tempn
               invent = Inv(stage1, tempn)
            End If
         End If
      Next k
      Inv(stage, j) = invent
   Next j
Next i
OPTF = F(1, 1)
OPTN = 1
OPTD = D(1, 1)
XS(1) = OPTD
nstate = NODE(1, OPTN)
For i = 2 To NSTAGE
   XS(i) = D(i, nstate)
   nstate = NODE(i, nstate)
Next i
            For i = 1 To NSTAGE : If (XS(i) <= 1) Then : XS(i) = 0 : Else : _XS(i) = XS(i) - 1 : End
            If : Next i
End Sub
```

```
Private Sub UpdateEnteringColumn()
    'Updates newly found column
    Dim i, j As Integer
    For i = 1 To NPart
      AR(i) = 0
      For j = 1 To NPart : AR(i) = AR(i) + BI(i, j) * XS(j) : Next j
    Next i
    If Verbose Then
      msg &= "--- Update New Column ---" & vbNewLine
      For i = 1 To NPart
        msg &= AR(i) & vbNewLine
      Next i
      msg &= vbNewLine
    End If
  End Sub


  Private Sub LeavingColumn()
    ' determines a dropping variable by ratio test
    Dim i As Integer
    Dim temp As Single
    NR = 0
    RMAX = 1.0E+30
    For i = 1 To NPart
      temp = 1.0E+30
```

```
    If AR(i) > 0 Then temp = StockusedByPattern(i) / AR(i)
    If (RMAX > temp) Then
       RMAX = temp
       NR = i
    End If
  Next
  If Verbose Then
    msg &= "--- Choose pattern to drop  {PIVOT ROW} ---" & vbNewLine
    msg &= "Pattern " & NR & " is leaving, at min ratio  " _
                & RMAX & vbNewLine
  End If
End Sub


Private Sub UpdateBInverse()
  ' Update B Inverse
  Dim i, j As Integer
  For i = 1 To NPart : ARINV(i) = -AR(i) / AR(NR) : Next i
  ARINV(NR) = 1 / AR(NR)
  For i = 1 To NPart
    If (i <> NR) Then
       For j = 1 To NPart : BI(i, j) = BI(i, j) + _
                ARINV(i) * BI(NR, j) : Next j
    End If
  Next i
```

```
    For j = 1 To NPart : BI(NR, j) = ARINV(NR) * BI(NR, j) : Next j
    If Verbose Then
        msg &= vbNewLine
        msg &= "---Update B Inverse ---" & vbNewLine
        Me.PrintMatrix(BI)
    End If
End Sub
Private Sub UpdateSolution()
    Dim i As Integer
    For i = 1 To NPart : StockusedByPattern(i) = _
                StockusedByPattern(i) - AR(i) * RMAX : Next i
    StockusedByPattern(NR) = RMAX
    ZB = ZB + CMIN1 * RMAX
    If Verbose Then
        msg &= vbNewLine
        msg &= "--- Update Solution ---" & vbNewLine
        msg &= "index  : basic variable solution" & vbNewLine
        For i = 1 To NPart
            msg &= i & "  :  " & StockusedByPattern(i) & vbNewLine
        Next i
        msg &= "The new cost is = " & ZB & vbNewLine
        'Me.PrintCurrentBasis(BI)
    End If
End Sub
```