



UNIVERSITÄT
HAMBURG



UNIVERSITÀ DEGLI
STUDI DELL'AQUILA



UNIVERSITAT AUTÒNOMA
DE BARCELONA

ERASMUS MUNDUS CONSORTIUM "MATHMODS"

Joint MSc Programme in
Mathematical Modelling in Engineering: Theory, Numerics, Applications

In the framework of the
Consortium Agreement and Award of a Joint/Multiple Degree 2013-2019

MASTER'S THESIS

An LP bound for 2-Dimensional Knapsack Problem by 1-Dimensional Cutting Stock Relaxation

Supervisor

Prof. Claudio Arbib

Candidate

Bikash Adhikari
Matricola : 249542

ACADEMIC YEAR 2017/2018

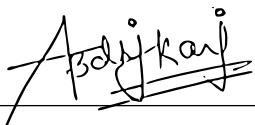
Laurea Magistrale in Ingegneria Matematica
Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica
Università degli Studi dell'Aquila
September 3, 2018

Declaration of Authorship

I, Bikash ADHIKARI, declare that this thesis titled, 'An LP bound for 2-Dimensional Knapsack Problem by 1-Dimensional Cutting Stock Relaxation' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:



Date:

UNIVERSITA DEGLI STUDI DELL'AQUILA

Abstract

Faculty Name

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Masters of Science

An LP bound for 2-Dimensional Knapsack Problem by 1-Dimensional Cutting Stock Relaxation

by Bikash ADHIKARI

The present work investigates the upper bound on the 2D KNAPSACK problem by 1D CUTTING STOCK relaxation. Given a set of small rectangles (the *items*), each one fitting into a big rectangle (the *bin*) and associated with a positive value, the 2D KNAPSACK problem calls for packing items into the bin for a total maximum value. To compute an upper bound on the maximum value, one can relax item connectivity: the items and the bin are sliced into strips of unit width, each strip regarded as a 1-dimensional object; the strips derived from the items (here called *pieces*) are then packed, up to availability, into individual strips derived from the bin, so that either all or none of the pieces from the same items are packed. In a solution, the pieces assigned to each strip form a pattern as in the well-known 1D CUTTING STOCK problem and we formulate an integer linear program to choose the patterns that maximize the total strip value. Finally, instead of finding an integer optimum (which is too demanding in computational terms), we solve the LP relaxation by column generation. A bound computed in this way has been evaluated and compared to optimum 2D packings for several problem instances found in the literature.

Keywords: 2-Dimensional Knapsack, 1-Dimensional Cutting Stock, Linear Program, Integer Linear Program, LP Relaxation, Upper Bound, column generation, Dantzig-Wolfe decomposition

Acknowledgements

Foremost, I would like to express my sincere gratitude to my advisor Prof. Claudio Arbib for his patient guidance, enthusiastic encouragement and useful critiques towards my thesis study and research. His willingness to give personal time despite busy schedule has been very much appreciated. I could not have imagined having a better advisor for my master thesis.

I would also like to thank the rest of the MathMods scientific committee: Prof. Bruno Rubino, Prof. Corrado Lattanzio, Prof. Marco Di Francesco and finally Cinzia Di Martino, Danilo Larivera & Astrid Benz, as the student coordinators, for supporting me through all the bureaucracy while I was staying in Italy and Germany.

Finally, I must express my very profound gratitude to my parents, flat-mates Pavan Kumar AV and Arun Pandey for providing me with unfailing support and continuous encouragement throughout my years of study, and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Contents

Declaration of Authorship	ii
Acknowledgements	v
List of Figures	viii
List of Tables	ix
1 Introduction	1
2 Mathematical Theory and Background	3
2.1 Complexity Classes	3
2.2 Linear and Integer Programming	6
2.2.1 Lagrangian Dual	6
2.2.2 The Simplex Method	8
2.3 Dantzig-Wolfe Decomposition	10
2.3.1 Column Generation	10
2.3.2 Dantzig-Wolfe Decomposition Principle	12
2.4 Dual-Feasible Functions	17
3 2-Dimensional Knapsack Problem	18
3.1 History and Overview of the Problem	20
3.2 Problem Description	21
4 Relaxation	24
4.1 One-Dimensional Cutting Stock Problem	24
4.2 One-Dimensional Cutting Stock Relaxation	25
4.3 Problem Formulation	27
4.4 Column Generation	28
4.4.1 Master Problem	28
4.4.2 Restricted Master Problem	29

4.4.3 Pricing Problem	31
4.5 ILP formulation for packing patterns into bin	35
4.6 Computational Results	37
5 Conclusions	40
Bibliography	43

List of Figures

2.1	Flowchart for reducibility of problems	4
2.3	Communication between Master Problem and Sub-Problem	12
4.1	Slicing of 2D item into 1D pieces	25
4.2	Example of a pattern	29
4.3	Initial set of patterns for solving the RMP	30
4.4	Patterns generated by column generation algorithm.	33
4.5	Flowchart: Column Generation	34
4.6	Stages of 2D packing by 1D relaxation	36
4.7	Comparision of Bounds obtained due to 1CS relaxation with the optimal value.	38

List of Tables

4.1	Set of 2D items that should be packed into the bin of size (10,10): size, minimum and maximum number of each item that is allowed to pack into the bin and their respective value.	26
4.2	Set of the 1D pieces obtained after slicing 2D items.	27
4.3	Set of patterns generated by the column generation algorithm	32
4.4	Computation Results	39

List of Abbreviations

LP	Linear Programming
ILP	Integer Linear Programming
MILP	Mixed Integer Linear Programming
BAB	Branch And Bound
BP	Bin Packing
CS	Cutting Stock
LP	Linear Programming
ILP	Integer Linear Programming
BLP	Binary Linear Programming
DFP	Dual Feasible Functions
MP	Master Problem
RMP	Restricted Master Problem
2BP	2-Dimensional Bin Packing
2SP	2-Dimensional Strip Packing
2CS	2-Dimensional Cutting Stock
2K	2-Dimensional Knapsack

Chapter 1

Introduction

The following thesis investigates the LP bound on two-dimensional Knapsack problem by one-dimensional Cutting Stock relaxation. This thesis can be roughly divided into three parts. The first part contains the theoretical background necessary to understand the relaxation and its implementation. The second part explains two-dimensional Knapsack problems and its variants and the third part explains the implementation of the relaxation to obtain the upper bound on the optimal solution.

Chapter 2 introduces the readers to the theory necessary to grasp the idea of the thesis. It starts with defining the complexity of the problems and covers the area of the linear program along with its duality theory and simplex method, a well-known algorithm for solving LP problems. Then we proceed with the introduction of Dantzig-Wolfe (DW) decomposition, which is an important tool for solving the large problems that cannot be solved using the standard simplex algorithm. This chapter also presents the Dual Feasible Functions (DFF), which provides the feasible dual bounds for the strong models which are close to the bounds obtained by column generation but without heavy computation burden.

Chapter 3 proceeds with the introduction of the two-dimensional Knapsack problem with their variants. It covers the background information about the problem, its mathematical formulation, and popular heuristics. We also present a review of the literature, relevant to the scope of this thesis. Chapter 4 describes the relaxation of the two-dimensional Knapsack problem into one-dimensional cutting stock. We present an example to explain the relaxation so that the reader can fully grasp the idea.

Chapter 5 presents the conclusion of the thesis and direction for the future research.

Note to the reader

The theory part of the thesis burrows from the book titled An Introduction to Continuous Optimization [4] and Introduction to Cutting and Packing Optimization: Problems, Modeling Approaches, Solution Methods [30]. The topic of dual feasible function was motivated by the paper from Alves et al. [3] on Dual-Feasible functions for Integer Programming and Combinatorial Optimization.

Chapter 2

Mathematical Theory and Background

In this chapter, we introduce the theory which is relevant for the development of the thesis. In section 2.1, we provide the discussion of complexity classes and section 2.2 briefly introduces the linear and integer programming, its dual and the widely known simplex method. Section 2.3 mainly focus on Dantzig-Wolf decomposition and Column Generation, which we believe is the proper framework to use in order to implement the thesis. Later, in section 2.4 we introduce the Dual-Feasible Functions which have been used to improve the resolution of a different combinatorial optimization problem with knapsack inequalities, including cutting and packing, scheduling and routing problems.

2.1 Complexity Classes

A Turing Machine is a mathematical model of computation that defines an abstract machine, which manipulates symbols on a strip of tape according to the table of rules. It works on an infinite memory tape divided into discrete cells. The machine positions its head over a cell and scans the symbol there. It has a finite number of states. The Deterministic Turing Machine (DTM), has a transition function that gives, for any given state and symbol the next symbol to be written, the direction where to move and the next state. DTM has a single computation path. The Non-Deterministic Turing Machine (NDTM) may have more than one action to be performed for any given situation. So, the computation of NDTM is a tree.

Class P and Class NP

In computational complexity, class P contains the set of decision problems or set of problems that can be solved by a deterministic turing machine in polynomial time. This means that the running time $T(n)$ of an algorithm is bounded by a polynomial

of input size (n) i.e. there exists an integer k such that $T(n)$ is $O(n^k)$. These are the class of easy to solve problems. Example: sorting, maximum matching.

The class NP contains a set of decision problems where *yes* instance can be decided in a polynomial time by deterministic computation. Notation NP stands for nondeterministic polynomial time. The formal definition of NP is a set of decision problems solvable in polynomial time by Non-Deterministic Turing Machine. Example: Bin Packing Problem, Travelling Salesman Problem.

The Class P and Class NP can be represented as P and NP, respectively. So, P is the class of easy to solve problems and NP is the class of easy to check problems. The class P is contained in NP i.e. $P \subseteq NP$.

Reducibility

Definition 2.1.1 *Problem A polynomially reduces to problem B, if polynomial time algorithm for B implies a polynomial time algorithm for A. A problem A is reducible to problem B if there is a function which converts an instance of A into an instance of B and solution of that instance provides a solution to the problem A.*

Let A and B be two decision problems. Let algorithm B^* solves problem B. If y is an input to B, then B^* will answer yes/no depending upon whether y belongs to B or not. So, reducibility is to find a transformation so that algorithm B^* can solve $x \in A$, $\Gamma(A) = B$. This conversion should take polynomial time.

Example: Travelling salesman polynomially reduces from the Hamiltonian circuit.

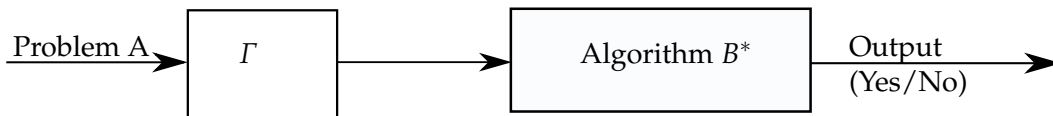


FIGURE 2.1: Flowchart for reducibility of problems

NP-Hard

Intuitively, these are the problems that are as hard as NP–Complete problems. Also, they do not have to be in NP and they do not need to be decision problems, it also includes search problems or optimization problems.

Definition 2.1.2 A problem X is NP-hard if for every problem $Y \in NP$, there is a polynomial time reduction from Y to X .

NP-Complete

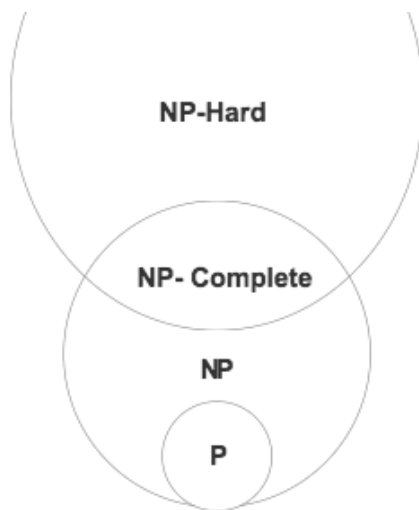
NP-Complete is a complexity class which represent set of all decision problems X in NP for which it is possible to reduce any other NP problems Y to X in polynomial time. They are the decision problems belonging to both NP and NP-hard complexity class.

Definition 2.1.3 A decision problem X is NP - Complete if,

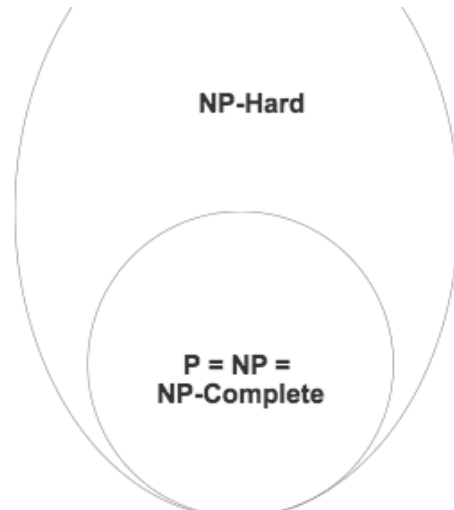
- $X \in NP$
- Every problem in NP is reducible to X in polynomial time.

Example: Boolean Satisfiability Problem, Graph Coloring Problem.

The importance of solving the NP-Complete problem is that if we are able to find an algorithm to solve NP-Complete in polynomial time then we can solve every other NP problem in polynomial time. For many NP problems, serious attempts are made to find the optimal solution in polynomial time but it appears to be intractable.



(A) $P \neq NP$



(B) $P = NP$

2.2 Linear and Integer Programming

Definition 2.2.1 Let $n, m \in \mathbb{N}$, $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, and $b \in \mathbb{R}^m$. The following optimization problem is called a **linear program** (LP).

$$\min_x \quad z_{LP} = c^T x \quad (2.1)$$

$$\text{s.t.} \quad Ax \geq b \quad (2.2)$$

$$x \in \mathbb{R}_+^n \quad (2.3)$$

where \mathbb{R}_+^n denotes non-negative real numbers.

Definition 2.2.2 Let $n, m \in \mathbb{N}$, $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, and $b \in \mathbb{R}^m$. The following optimization problem is called an **integer program** (IP).

$$\min_x \quad z_{IP} = c^T x \quad (2.4)$$

$$\text{s.t.} \quad Ax \geq b \quad (2.5)$$

$$x \in \mathbb{Z}_+^n \quad (2.6)$$

where \mathbb{Z}_+^n denotes non-negative integer numbers.

Definition 2.2.3 A set $S \subseteq \mathbb{R}^n$ is **convex** if for any $x, y \in S$, and any $\lambda \in [0, 1]$, we have $\lambda x + (1 - \lambda)y \in S$.

Definition 2.2.4 Given a set $S \subseteq \mathbb{R}^n$, the point $X \in \mathbb{R}^n$ is a **convex combination** of points of S if there exists a finite set of n points of X in S and $\lambda \in \mathbb{R}_+^n$ with $\sum_{i=1}^n \lambda_i = 1$ and $X = \sum_{i=1}^n \lambda_i x_i$.

Definition 2.2.5 Given a set $S \subseteq \mathbb{R}^n$, the set of all points that are the convex combination of S is called the **convex hull** of S and is denoted by $\text{conv}(S)$.

2.2.1 Lagrangian Dual

LP duality is the important concept in linear programming. Consider the linear program (2.1-2.3). We refer to this problem as Primal Problem. For any arbitrary vector $\lambda \in \mathbb{R}^m$ we define the Lagrangian function as:

$$L(x, \lambda) = c^T x + \lambda^T (b - Ax) \quad (2.7)$$

Further, we define a Lagrangian dual function as

$$\theta_{LP}(\lambda) = \min_{x \geq 0} L(x, \lambda) = \min_{x \geq 0} \{c^T x + \lambda^T (b - Ax)\} = b^T \lambda + \min_{x \geq 0} \{(c^T - \lambda^T A)x\} \quad (2.8)$$

For any $\lambda \geq 0$, the Lagrangian dual function is the Lagrangian relaxation of the primal problem and its optimal solution in x yields the lower bound on the optimal solution of a primal problem.

Assume x^* solves (2.1-2.3) with optimal objective function value $z_{LP}^* = c^T x^*$ and let \bar{x}_λ solves the minimization problem (2.8). From equation (2.2) we have that $b - Ax \leq 0$ and hence we can derive the following bounds

$$\begin{aligned} \theta_{LP}(\lambda) = L(\bar{x}_\lambda, \lambda) &= b^T \lambda + (c^T - \lambda^T A)\bar{x}_\lambda \leq c^T x^* + \lambda^T (b - Ax^*) \\ &= z_{LP}^* + \lambda^T (b - Ax^*) \leq z_{LP}^* \end{aligned} \quad (2.9)$$

Thus for any $\lambda \geq 0$, via the optimal solution \bar{x}_λ to (2.8), $L(\bar{x}_\lambda, \lambda)$ provides the lower bound to the primal objective function value z_{LP}^* .

We now define a Lagrangian dual problem where we need to find the non-negative value of λ for which $\min_{x \geq 0} L(\bar{x}_\lambda, \lambda)$ yields the tightest bound on z_{LP}^* .

$$\begin{aligned} \max_{\lambda \geq 0} \quad & \theta_{LP}(\lambda) = b^T \lambda + \min_{x \geq 0} \{(c^T - \lambda^T A)x\} \\ \text{s.t.} \quad & \lambda \geq 0 \end{aligned} \quad (2.10)$$

However, if any component of the vector $(c^T - \lambda^T A)x$ is negative, the minimization term in the objective of (2.10) will go to $-\infty$ and such a solution will never be optimal in the maximization problem with regard to λ . This motivates the inclusion of additional constraint $(c^T - \lambda^T A)x \geq 0$ in which case the second term in the objective function vanishes and we arrive at the dual of the linear programming dual of (2.1-2.3),

$$\max_{\lambda} \quad \theta_{LP} = b^T \lambda \quad (2.11)$$

$$\text{s.t.} \quad A^T \lambda \leq c \quad (2.12)$$

$$\lambda \geq 0 \quad (2.13)$$

We refer to the decision variable $\lambda \in \mathbb{R}^m$ of the dual problem as the dual vector of the primal problem. This dual vector plays an important role in the simplex method and the column generation process. First, we present the necessary and sufficient conditions for the global optimality of linear programs, as stated in [4].

Theorem 2.2.1 (Necessary and Sufficient condition for the global optimality of LP) For $x \in \mathbb{R}^n$ to be an optimal solution to the linear program (2.1-2.3), it is both necessary and sufficient that following holds,

- x is a feasible solution to (2.1-2.3).
- $\lambda \in \mathbb{R}^m$ is a feasible solution to (2.11-2.13) and
- *Complementary Slackness Condition: The primal-dual pair (x, λ) satisfies the complementary slackness conditions,*

$$\begin{aligned} (A_i x - b_i) \lambda_i &= 0 & \forall i \\ (c_j - \lambda^T A_{.j}) x_j &= 0 & \forall j \end{aligned} \quad (2.14)$$

where A_i is the i^{th} row of the constraint matrix A and $A_{.j}$ is the j^{th} column of A .

2.2.2 The Simplex Method

Consider the LP in standard form,

$$\min_x \quad z_{LP} = c^T x \quad (2.15)$$

$$\text{s.t.} \quad Ax = b \quad (2.16)$$

$$x \geq 0 \quad (2.17)$$

where $A \in \mathbb{R}^{m \times n}$, $c \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$. We know that feasible region of the linear program is a polyhedron. The representation theorem and the theorem regarding the existence [4] states that if there is a finite optimal solution to (2.15-2.17), then there exists an optimal solution among the extreme points of the polyhedron describing the feasible region of the problem. The theorems are stated as follows,

Theorem 2.2.2 (Representation Theorem) Let $X = \{x : Ax \geq b, x \geq 0\}$ and $P = \{x_1, x_2, \dots, x_p\}$ be the set of extreme points of X . Further, let $C = \{x \in \mathbb{R}^n | Ax = 0, x \geq 0\}$ and $R = \{y_1, y_2, \dots, y_r\}$ be the sets of extreme rays of C .

- X is non-empty and bounded if and only if P is non-empty (and finite) and R is empty.
- X is unbounded if and only if R is nonempty (and finite).

- Every point $x \in X$ can be expressed as the convex combination of extreme points in P and non-negative linear combination of the points in R , i.e.,

$$\begin{aligned} x &= \sum_p \lambda_p x_p + \sum_r \mu_r y_r \\ \sum_p \lambda_p &= 1 \\ \lambda_p &\geq 0, \quad \mu_r \geq 0 \end{aligned} \tag{2.18}$$

Theorem 2.2.3 (Existence and properties of an optimal solution) Let $X = \{x : Ax \geq b, x \geq 0\}$, P is set of extreme points of X and R is the set of extreme rays of X and consider the linear program,

$$\begin{aligned} \min_x \quad & z = c^T x \\ \text{s.t.} \quad & x \in X \end{aligned} \tag{2.19}$$

- This problem has a finite optimal solution if and only if X is nonempty and z is lower bounded on X , that is, if X is nonempty and $c^T y_r \geq 0, \forall y_r \in R$.
- If the problem has a finite optimal solution, then there exists an optimal solution among the extreme points.

At each iteration simplex algorithm starts at the current basic feasible solution (BFS) and moves to the adjacent BFS such that the objective value decreases. The principle of the simplex algorithm is to move to an adjacent extreme point such that the objective function decreases. Hence the geometric notion of the extreme points are represented by the algebraic notion of BFS and moving from one BFS to another implies moving from one extreme point to the other. In view of the LP problem (2.15-2.17), these BFSs are constructed by letting $n - m$ variables equal zero— non-basic variable— and letting the remaining m basic variables construct the BFS at hand, i.e. the extreme point we want to represent. Search for the non-basic variable is realized by using the concept of negative reduced cost of a variable. The reduced cost can be said to be a measure for the cost—with regard to the objective function—to increase the variable by a small amount. Hence for the minimization problem, we wish to find the non—basic variable with the negative reduced cost to enter the basis. Conversely, for the maximization problem, we search for the non—basic variables with the non-negative reduced cost.

2.3 Dantzig-Wolfe Decomposition

Generally, there are different ways of modelling an IP problem. Any model that provides a set of feasible integer solutions is valid. However, there may be differences in LP relaxation. Some models may provide a closer description of the set of the feasible integer solutions. Let $X_1 \subseteq \mathbb{R}^n$ and $X_2 \subseteq \mathbb{R}^n$ be the convex sets of the solution of the LP relaxation of two valid IP models. A model with a set X_1 is said to be stronger than the model with a set X_2 , if $X_1 \subset X_2$. Let X_{IP} be the set of solutions of IP model. The convex hull of X_{IP} denoted by $conv(X_{IP})$ is the strongest model for the IP problem. Using LP relaxation, the solution would never be fractional because the extreme points are all integer. But, the difficulty arises in assigning all constraints that need to define the convex hull, $conv(X_{IP})$. So, one has to resort to the alternatives that are weaker than $conv(X_{IP})$, but may be stronger than the LP relaxation.

In this section, we introduce the general idea of Column Generation (CG) followed by an overview of Dantzig-Wolfe decomposition principle.

2.3.1 Column Generation

Column Generation[8], [20] is a classical solution approach to solve a mathematical program by iteratively adding the variables of the model. The idea is to decompose this problem so that the problem has to be never solved with all the subproblems. Instead, a master problem is devised and solved while the subproblems are added individually. As a result of the only series of smaller problems needed to be solved. Consider a linear program,

$$\min_x \quad z_{MP} = \sum_{j \in J} c_j x_j \quad (2.20)$$

$$\text{s.t.} \quad \sum_{j \in J} a_{ij} x_j \geq b_i \quad \forall \quad i \in I \quad (2.21)$$

$$x_j \geq 0, \quad j \in J \quad (2.22)$$

where $a_{ij} \in \mathbb{R}^{m \times n}$, $b_i \in \mathbb{R}^n$, $c_j, x_j \in \mathbb{R}^m$ for all J and $|J| = m$ is the number of variables and $|I| = n$ is the number of constraints. We will refer to this problem as Master Problem (MP) with an optimal objective function value z_{MP}^* . While in many applications we have an exponential number of variables compared to the number

of constraints which makes the problem (2.20-2.22) so enormous and explicitly enumerating them all may be computationally intractable.

Instead, we consider the Restricted Master Problem (RMP) which contains only $J' \subseteq J$ variables and evaluate reduced costs only by implicit enumeration.

$$\min \sum_{j \in J'} c_j x_j \quad (2.23)$$

$$\text{s.t.} \quad \sum_{j \in J'} a_{ij} x_j \geq b_i \quad \forall i \in I \quad (2.24)$$

$$x_j \geq 0, \quad j \in J' \quad (2.25)$$

We solve this RMP to optimality, let z_{RMP}^* be the optimal objective function value. The optimal objective function value to the RMP gives an upper bound on the MP optimal objective function value i.e. $z_{MP}^* \leq z_{RMP}^*$. However, the optimal solution for the RMP needs not to be the optimal solution for the MP. Let π be the dual variable. Then the dual of the RMP is as follows,

$$\max_{\pi} \quad \sum \pi_i b_i \quad (2.26)$$

$$\text{s.t.} \quad \sum_{i \in I} a_{ji} \pi_i \leq c_j \quad \forall j \in J' \quad (2.27)$$

$$\pi_i \geq 0 \quad (2.28)$$

With these definitions, MP is identically $MP(I)$ and $D(I)$ is the dual of the MP. The column generation algorithm rests on the following simple property.

Property 1: Let I' be a subset of I and π^* the optimal of $D(I')$. If π^* is feasible for $D(I)$, π^* is optimal for $D(I)$.

We know that the simplex method gives a basic feasible solution to both primal and dual problem that satisfy complementary slackness conditions. After optimal solution to RMP is found we then have to check if this new variable added to our model results in negative reduced costs. It is then convenient to let this variable enter the basis. If a new pattern is available, one can decide whether this pattern should be used or not by proceeding as above.

However, the problem is to find these variables which yield in negative reduced cost. Not only the number of possible patterns is exponentially large but the patterns

are also not known explicitly. In order to see whether the variable with a negative reduced cost exists it is sufficient to solve the following optimization problem.

$$\bar{c}^* = \min_{j \in J} \left\{ c_j - \sum_{i \in I} \pi_i^* a_{ij} \right\}. \quad (2.29)$$

If $\bar{c}^* \leq 0$, the variables x_{j^*} and its coefficients $[c_{j^*}, a_{ji^*}]^T$ are added to the RMP and this is solved to optimality to obtain the dual variable and the process is repeated until no further improving variable is found i.e. $\bar{c}^* \geq 0$. The problem (2.29) is often called Pricing Problem (PP)[13]. Figure 2.3 shows the communication between the MP and pricing sub-problem [24]. The process is initialized with an artificial, a heuristic, or a previous solution. When dealing with some finite set $a_i \in A$, which is always true in practice, the column generation algorithm is exact. Let z_{RMP}^* denote the optimal objective function value to RMP then we have an upper bound on MP optimal objective function value z_{MP}^* in each iteration. When an upper bound $\kappa \geq \sum_{i \in I} x_i$ holds for the optimal solution of the master problem then we also have a lower bound. We cannot reduce z_{MP}^* by more than κ time the smallest reduced cost $\bar{c}^* : z_{RMP}^* + \kappa \bar{c}^* \leq z_{MP}^* \leq z_{RMP}^*$.

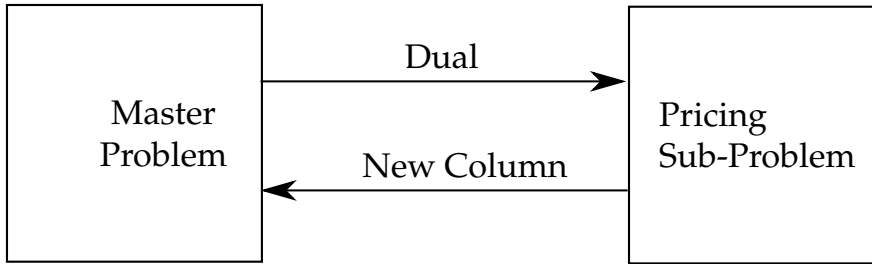


FIGURE 2.3: Communication between Master Problem and Sub-Problem

2.3.2 Dantzig-Wolfe Decomposition Principle

Dantzig-Wolfe decomposition is an important tool to solve the large structured model that cannot be solved by using the standard simplex algorithm. This decomposition principle is about the changing the representation from the set of constraints to set

of extreme points. Consider a linear program,

$$\min_x \quad z_{LP} = c^T x \quad (2.30)$$

$$\text{s.t.} \quad Ax \geq b \quad (2.31)$$

$$Dx \geq d \quad (2.32)$$

$$x \geq 0 \quad (2.33)$$

where $x, c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $d \in \mathbb{R}^p$, $A \in \mathbb{R}^{m \times n}$, $D \in \mathbb{R}^{p \times n}$. Consider the polyhedron $X = \{x : Dx \geq d\}$. Let z_{LP}^* be the optimal objective value function to the LP, replacing the polyhedron X by $\text{conv}(X)$ does not change the value of z_{LP}^* . Hence, the MP is characterized by the equations:

$$\min \quad c^T x \quad (2.34)$$

$$\text{s.t.} \quad Ax \geq b \quad (2.35)$$

$$x \in \text{conv}(X) \quad (2.36)$$

By Minkowski's Representation theorem, we can write any $x \in X$ as a finite convex combination of extreme points $\{x_p\}_{p \in P}$ and non-negative combination of extreme rays $\{y_r\}_{r \in R}$ of $\text{conv}(X)$, i.e.,

$$x = \sum_{p \in P} c^T x_p \lambda_p + \sum_{r \in R} c^T y_r \mu_r \quad (2.37)$$

for some $\lambda_p \geq 0$, $p \in P$, such that $\sum_{p \in P} \lambda_p = 1$ and $\mu_r \geq 0$, $r \in R$. Now, substituting the value of x from (2.18) into equation (2.34-2.36), and from theorem 2.2.1, we get the following reformulation which is solved by column generation.

$$\min_{\lambda, \mu} \quad \sum_{p \in P} c^T x_p \lambda_p + \sum_{r \in R} c^T y_r \mu_r \quad (2.38)$$

$$\text{s.t.} \quad \sum_{p \in P} \lambda_p A x_p + \sum_{r \in R} \mu_r A y_r \geq b \quad (2.39)$$

$$\sum_{p \in P} \lambda_p = 1 \quad (2.40)$$

$$\lambda_p \geq 0 \quad \forall p \in P \quad (2.41)$$

$$\mu_r \geq 0 \quad \forall r \in R \quad (2.42)$$

We consider only a subset $P' \subset P$ and $R' \subset R$ of the MP variables to construct a RMP. Let $\bar{\pi}$ and $\bar{\lambda}$ be the dual optimal solution to the RMP, obtained from the equation (2.38-2.42), where the variable $\bar{\lambda}$ corresponds to the convexity constraint. Then we need to check for the negative reduced cost. Referring to the previous linear formulation (2.38-2.42), this results in solving the following linear program.

$$\min_{x \in P \cup R} \{ (c^T - \bar{\pi}^T A) x - \bar{\lambda} \mid Dx \geq d, \quad x \geq 0 \} \quad (2.43)$$

So, when the minimum is negative and finite, an optimal solution to the equation (2.43) is an extreme point of $\text{conv}(X)$ and we add a variable with the coefficient column $[cx_j, Ax_j, 1]$ to the RMP. However, when the minimum is minus infinity then the solution obtained is the extreme ray of $\text{conv}(X)$ and we add the column $[cx_r, Ax_r, 0]$.

However, if we assume the polyhedron X to be bounded then from Theorem 2.2.2 the MP reduces to

$$\min_{\lambda} \sum_{p \in P} c^T x_p \lambda_p \quad (2.44)$$

$$\text{s.t.} \quad \sum_{p \in P} \lambda_p Ax_p \geq b \quad (2.45)$$

$$\sum_{p \in P} \lambda_p = 1 \quad (2.46)$$

$$\lambda_p \geq 0 \quad \forall p \in P \quad (2.47)$$

and the pricing problem reduces to,

$$\min_{x \in P} \{ (c^T - \bar{\pi}^T A) x - \bar{\lambda}, x \geq 0 \} \quad (2.48)$$

Now, we proceed to a special case of DW decomposition, where the constraint matrix A has a special structure.

Block-Diagonal Models

We consider the case when the constraint matrix A has the block-diagonal structure. The usefulness of the DW decomposition apparently becomes relevant when the corresponding constraint matrix has a block-diagonal structure. Consider the linear

program,

$$\begin{aligned}
 & \min \quad c^T x \\
 & \text{s.t.} \quad Ax \geq b \\
 & \quad \quad Dx \geq d \\
 & \quad \quad x \geq 0
 \end{aligned} \tag{2.49}$$

where A , has a special structure.

$$Ax = \begin{bmatrix} A_1 & A_2 & \cdots & A_K \\ D_1 & & & \\ & D_2 & & \\ & & \ddots & \\ & & & D_K \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_K \end{bmatrix} \geq \begin{bmatrix} b \\ d_1 \\ \vdots \\ d_K \end{bmatrix} \tag{2.50}$$

The constraints

$$\sum_i A_i x_i \geq b \tag{2.51}$$

corresponding to the top row of the submatrix are called the *coupling constraints*. Each $X_i = \{D_i x_i \geq d_i, x_i \geq 0\}$, $i = 1, \dots, K$ gives rise the representation in (2.37). The decomposition yields each K subproblems, each with its own convexity constraint and the dual variables λ'_k which leads to the following pricing problem.

$$\min \left\{ (c_k^T - \bar{\pi}^T A_k) x_k - \bar{\lambda}_k, x_k \geq 0 \right\} \tag{2.52}$$

Extension to Integer Programming

In this section, we will extend the ideas of Dantzig-Wolfe decomposition and column generation, to integer linear programming [31]. We again consider the linear program (2.30-2.33) with additional integrality constraint in the decision variable, yielding the following ILP,

$$\min \quad c^T x \tag{2.53}$$

$$\text{s.t.} \quad Ax \geq b \tag{2.54}$$

$$Dx \geq d \tag{2.55}$$

$$x \geq 0, \quad \text{integer} \tag{2.56}$$

Let $X = \{x \in \mathbb{R}^n | Dx \geq d\}$ and $V = X \cap \mathbb{Z}^+$. We will assume that V is bounded. Now, the decomposition for integer programming begins by stating a result stated in [28], regarding the representation of the points in space constructed by the intersection of linear (convex) region and an integer space.

Theorem 2.3.1 Representation theorem for integer linear programming

Let $X = \{x \in \mathbb{R}^n | Dx \geq d, x \geq 0\} \neq \emptyset$ and $V = X \cap \mathbb{Z}^+ \neq \emptyset$. Then there exists a finite set of integer points $\{x_p\}_{p \in P} \subseteq V$ and finite set of integer rays $\{y_r\}_{r \in R}$ of R such that

$$V = \left\{ x \in \mathbb{R}_+^n \mid x = \sum_{p \in P} x_p \lambda_p + \sum_{r \in R} y_r \mu_r, \sum_{p \in P} \lambda_p = 1, \lambda_+^P, \mu_+^R \right\} \quad (2.57)$$

Since we have assumed that V is bounded so, the second term of the element x in X , representing the extreme rays $\{y_r\}_{r \in R}$ vanishes. So, we reached the trivial result that any point $x \in V$ can be represented as one of the integer points in V . While substituting the value of x in equation (2.53-2.56) we get the following ILP,

$$\min \quad \sum_{p \in P} c^T x_p \lambda_p \quad (2.58)$$

$$\text{s.t.} \quad \sum_{p \in P} A x_p \lambda_p \geq b \quad (2.59)$$

$$\sum_{p \in P} \lambda_p = 1 \quad (2.60)$$

$$\lambda_p \in \mathbb{Z}_+, \quad p \in P \quad (2.61)$$

We perform the continuous relaxation of the integer requirement, reaching the linear MP, similar to general Dantzing-Wolfe MP in (2.30-2.33), but the difference is that in the later MP, $\{x_p\}_{p \in P}$ represent the integer points in V , which may be inner points as well as extreme points to the polyhedron X , whereas in pure linear case, they represent the extreme points of the feasible region polyhedron X .

As discussed in the previous section, we consider only a subset $P' \subset P$ to obtain an RMP. With the dual variables $\bar{\pi}$ and $\bar{\lambda}$ obtained from solving the RMP, obtained from equation (2.58-2.61) we solve following pricing subproblem to generate columns to RMP.

$$\min \quad \{(c^T - \bar{\pi}^T A) x - \bar{\lambda} \mid Dx \geq d, \quad x \geq 0\} \quad (2.62)$$

The subproblem need not always solved to optimality but can be used to generate the columns with negative reduced cost, when possible. When no more improving columns can be added, then the column generation process is terminated.

2.4 Dual-Feasible Functions

The DW-decomposition is one technique to obtain stronger models. In any case, the model that results from DW-decomposition has an exponential number of decision variables and it is not possible to enumerate them all. One may turn to column generation, which has been fruitful with IP problems but they are complex, time-consuming to implement and may still represent a substantial computational burden.

Dual Feasible Functions (DFF) [3] provide feasible dual solutions of strong models whose corresponding lower bounds are often very close to the lower bounds provided by column generation models. As DFF can be computed quickly, the computational burden can be small when compared to column generation algorithms. Furthermore, for a given problem, it is often possible to derive not only a single DFF but several DFF, or even families of different DFF, each providing a different dual feasible solution. By using several DFF, we aim at obtaining at least a feasible dual solution that provides a decent lower bound.

Definition 2.4.1 *A function $f : [0, 1] \rightarrow [0, 1]$ is a dual-feasible function, if for any finite index set I of non-negative real numbers $x_i \in \mathbb{R}_+, i \in I$, it holds that*

$$\sum_{i \in I} x_i \leq 1 \implies \sum_{i \in I} f(x_i) \leq 1.$$

Example: $f(0) = 0$ and $f(x) \leq 1/\lfloor 1/x \rfloor$ for all $x \in [0, 1]$.

Dual-feasible functions are generally defined in $[0, 1]$. However, using discrete values instead may lead to simpler formulations namely when the original input data is an integer. This alternative formulation yields a so-called discrete dual-feasible function whose definition is given next.

Definition 2.4.2 *A discrete dual-feasible function $f : \{0, 1, \dots, d\} \rightarrow \{0, 1, \dots, d'\}$ with $d, d' \in \mathbb{N} \setminus \{0\}$ is such that*

$$\sum_{i \in I} x_i \leq d \implies \sum_{i \in I} f(x_i) \leq f(d) = d'$$

for any finite index set I of non-negative integer numbers $x_i \in \mathbb{N}, i \in I$.

Chapter 3

2-Dimensional Knapsack Problem

In industrial processes as well as logistics applications, it is often required to allocate small rectangular items to larger rectangular items with a minimum waste. For example, in textile or paper manufacturing, a standardized roll of cloths or cardboard are given and one needs to obtain certain required items by using a minimum roll length. In many applications, standard stock units are generally large rectangular sheets of one or few different sizes and the goal is to minimize the total area of the sheets necessary to produce given amounts of small rectangles of given sizes.

In literature, the relevant optimization models are referred to as two-dimensional KNAPSACK (2K) or CUTTING STOCK (2CS) problems, the difference lies in the number of each small item required (one or few units in the former case and many in the latter). In fact, those models extend well beyond stock cutting applications and include task scheduling, VLSI design, image processing, packing goods in crates, commercials assignment to TV breaks, truck loading, just to name a few. Let us consider the set of n rectangular items with width, height, and positive value. The item is to be cut/packed with the edge parallel to the edges of a rectangular bin. Based on the application, there are many variations of knapsack problems. Let us consider the set of n items $i \in I = \{1, 2, \dots, n\}$ with the width w_i , height h_i and positive value p_i . The objective is to maximize the total sum of the value of items packed into the bin while the maximum total area of the chosen items should not exceed the total area of bin WH .

- **Bounded Knapsack Problem:**

In the bounded knapsack problem, for each item $i \in I$ there is an upper bound and lower bound (which is a positive integer). So, the maximum and the minimum number of times each item can be selected to put on the knapsack is bounded.

- **Unbounded Knapsack Problem**
The unbounded knapsack problem does not put any bound on the number of times each item can be selected.
- **Multiple Knapsack Problem**
In this class of knapsack problem, the items are subdivided into k class denoted by I_k and the multiple knapsack problem requires to choose exactly one item from each class.
- **Quadratic Knapsack Problem**
The quadratic knapsack problem requires to maximize the quadratic objective functions subjected to binary or linear capacity constraints.
- **Subset-sum Problem**
In subset-sum problems, the weight of an item is equal to its profit.

In this thesis, we focus on evaluating the upper bound on the optimal solution of the 2-Dimensional Bounded Knapsack (2K) problem.

ρ -Approximation Algorithm

The primary goal of the knapsack problem is to maximize the total value of items packed into the bin while the total area of the chosen items should not exceed the total area of the bin. The knapsack problem is an NP-Hard [18] thus finding an exact solution for any given input is possible only in exponential time. Hence, finding an approximation algorithm will be a reasonable approach to cope with the NP-Hardness of the problem. The approximation algorithms are guaranteed to run in polynomial time and to find a good quality solution.

For a given optimization problem P , there exists an algorithm A such that for any instances I of problem P , it gives the solution $A(I)$, we say that the algorithm A provides an ρ -approximation solution for a problem P when for any instances I , $1/\rho \leq O(I)/A(I) \leq \rho$. Here, $O(I)$ is the optimal solution for instance I of problem P and algorithm A is called the ρ -approximate algorithm.

Let P is an NP-Hard optimization problem and A is an approximation scheme for the problem P for every $\rho > 0$ which returns the solution P_{sol} such that $P_{sol} \leq (1 + \rho)P_{opt}$, for a minimization problem with an optimal value P_{opt} . The algorithm A is called the Polynomial-Time Approximation Scheme (PTAS) if it runs in a polynomial time of n and as we decrease ρ the running time of the algorithm increases drastically. The set of problems that allows PTAS are called class PTAS problems.

If the algorithm A is polynomial in both problem size n and $1/\rho$, it is called Fully Polynomial-Time Approximation Scheme (FPTAS) [27].

3.1 History and Overview of the Problem

Gilmore and Gomory [8], [20] introduced, for this class of problems, the concept of column generation, by deriving it from earlier ideas of Ford and Fulkerson [26], and Dantzig and Wolfe [12]. An unconstrained two-dimensional cutting stock problem, no constraint on number of pieces of same items that are cut, is solved using dynamic programming by Gilmore and Gomory [19] and with recursive tree search procedure by Hertz [22]. The latter provided high computational speeds in the solution compared to the former. Christofides and Whitlock [11] have presented the tree-search algorithm for the solution of two-dimensional constrained guillotine cutting stock problem that incorporates both dynamic programming and a transportation routine into a tree search algorithm. Wang [32] and Oliveira and Ferreira [29] provided heuristic approaches for the constrained guillotine cutting stock problem.

Beasley [5] provided the tree search procedure for unconstrained two-dimensional guillotine cutting problem. The provided model used Lagrangian relaxation and subgradient optimization for the upper bound. Christofides and Hadjiconstantinou [10] provided the improvement on the algorithm developed by Christofides and Whitlock [11], which limits the size of the tree search by using the tighter upper bound derived from the state space relaxation of a dynamic programming formulation of the original problem. A completely different graph-theoretical characterization of packing of a set of items into a single has been proposed by Fekete and Schepers [15]. Based on the feasibility of the packing, it is proved that the characterization can be easily extended to higher dimensions. Also, Bortfeldt and Tobias[7] proposed a genetic algorithm for 2K with rectangular pieces.

The upper bound in Belov et al. [6] is obtained by relaxing item connectivity, so transforming each d -dimensional object ($d \geq 2$) into many 1D strips of various widths w_i . Packing strips become then a 1CS-like problem with limited bin availability. Bound is enforced by requiring contiguity of bins containing strips from the same item. Lower and upper bounds of 2CS or 2BP are associated with relaxations and primal solutions, respectively. Dual Feasible Functions (DFF) [14] are introduced to determine lower bounds to CS or BP more quickly than solving the well known Gilmore-Gomory LP [8],[20] by column generation.

Approximation Algorithms

The research on approximation algorithms of 2K has started quite recently. Caprara and Monaci [9] addressed the problem on 2K which aimed at packing the maximum-valued subset of rectangles from the given set into the big rectangle and proved the worst-case performance of the associated upper bound by 1K relaxation and weight of each item equal to its area. Jansen and Zhang [23] proposed $(2 + \varepsilon)$ -approximation algorithm for 2K which allowed rotation and packing square also. Fishkin, Gerber, Jansen and Solis-Oba [16] and Han, Iwama and Zhang [21] independently presented PTAS for packing all squares and the profits equal to their areas. Kulik and Shachnai[25] proved that there exists no EPTAS for 2K.

3.2 Problem Description

A very important problem both if considered stand-alone and when arising from Dantzig-Wolfe decomposition of 2K aims at maximizing the total value of the items packed into a single bin of given size. Namely, consider n rectangular items R_i , $i \in I$, $|I| = n$, of width w_i and height h_i , each associated with a real positive value p_i to be packed into a large stock rectangle R_0 of width W and height H (the bin). Without loss of generality, assume both edges of each R_i specified as positive integers and $w_i \leq W$, $h_i \leq H$ for $i = 0, 1, \dots, n$. Assume also that the R_i 's must be packed into R_0 with their edges parallel to those of R_0 and rotation (that is, interchanging w_i and h_i) is not allowed. We want to

Pack items into the bin such that the total value of the items packed is maximum.

The Beasley-Type Model

In Beasley-type model, decision variables are used to indicate the allocation point (bottom-left corner) (x_i, y_i) of piece $i \in I$. In order to keep the number of binary variables small, we consider the sets of possible allocation coordinates

$$S(w, W) = \{r_0, \dots, r_\alpha\} = \{r : r = \sum_{i \in I} w_i a_i \leq W, \quad a_i \in \{0, 1\}, \quad i \in I\}$$

with $0 = r_0 < r_1 < \dots < r_\alpha \leq W$ and

$$S(h, H) = \{s_0, \dots, s_\beta\} = \{s : s = \sum_{i \in I} h_i a_i \leq H, \quad a_i \in \{0, 1\}, \quad i \in I\}$$

with $0 = s_0 < s_1 < \dots < s_\beta \leq H$. Let $J = \{0, \dots, \alpha - 1\}$ and $K = \{0, \dots, \beta - 1\}$ denotes the corresponding index sets, respectively. The region covered by rectangle i , if it gets the allocation point (x_i, y_i) , is denoted by

$$R_i(x_i, y_i) = \{(x, y) \in \mathbb{R}^2 : x_i \leq x < x_i + w_i, \quad y_i \leq y < y_i + h_i\}.$$

Based on the allocation points (x_i, y_i) for $i \in I' \subseteq I$, we define the corresponding the arrangement or pattern

$$A(I') = \{R_i(x_i, y_i) : i \in I'\}.$$

According to the approach suggested by Beasley [5] the binary decision variables $\forall i \in I, \forall p \in J_i = \{j \in J : r_j \leq W - w_i\}$, and $\forall q \in K_i = \{k \in K : s_k \leq H - h_i\}$ are as follows:

$$x_{ipq} = \begin{cases} 1, & \text{if item } i \text{ has allocation point } (x_i, y_i) = (r_p, s_q) \\ 0, & \text{otherwise} \end{cases}$$

In difference to [5] where each point (p, q) with $p \in \{0, \dots, W - 1\}$ and $q \in \{0, \dots, H - 1\}$ could be a potential allocation point, Scheithauer [30] applied a possible reduction of the model size which is caused by considering only appropriate potential coordinates. The patterns whose allocation points belonged to $S(w, W) \times S(h, H)$ were called *normalized*. The bottom-left justified in the [5] are normalized. A bottom-left justified pattern is obtained if all placed items cannot be shifted down- or leftwards. In [5] the non-overlapping condition of places pieces is formulated using coefficients.

$$a_{ipqjk} = \begin{cases} 1, & \text{if } x_{ipq} = 1, (r_j, s_k) \in R_i(r_p, s_q) \\ 0, & \text{otherwise} \end{cases}$$

$i \in I, p, j \in J, q, k \in K$, indicating that point (r_j, s_k) is covered if item i is allocated at (r_p, s_q) or not. Then the program is

$$\max \quad \sum_{i \in I} p_i \sum_{p \in J_i} \sum_{q \in K_i} x_{ipq} \quad (3.1)$$

$$\text{s.t.} \quad \sum_{i \in I} \sum_{p \in J_i} \sum_{q \in K_i} a_{ipqjk} x_{ipq} \leq 1, \quad j \in J, \quad k \in K, \quad (3.2)$$

$$\sum_{p \in J_i} \sum_{q \in K_i} x_{ipq} \leq 1, \quad i \in I \quad (3.3)$$

$$x_{ipq} \in \{0, 1\}, \quad i \in I, \quad p \in J_i, \quad q \in K_i. \quad (3.4)$$

Constraint 3.2 ensures that at most one piece covers point (r_j, s_k) because of constraint 3.3 which that any piece is placed at most once.

This two-dimensional Knapsack problem is recognized as NP-Hard, as it is a generalization of the ordinary (1D) knapsack problem [18]. This implies that a polynomial time algorithm for this problem cannot be found unless $P = NP$ or, saying it differently, that the best algorithm known so far to find an optimal solution requires, in the worst case, a number of steps exponential in the size of the problem. Due to problem complexity, finding good upper and lower bounds to the value of optimal solutions can, therefore, be crucial in order to improve the practical efficiency of exact solution algorithms. We proceed to the Chapter 4, to investigate the upper bound on the optimal value of two-dimensional knapsack problem by relaxing into the one-dimensional cutting stock problem.

Chapter 4

Relaxation

In this chapter, we implement the relaxation to compute an upper bound on the optimal solution of the two-dimensional KNAPSACK (2K). In section 4.1, we introduce 1CS problem and the similarities in cutting and packing. Section 4.2 proceeds by explaining the one-dimensional cutting stock relaxation with the help of an example. In section 4.4, we implement the column generation algorithm to pack the 1D pieces into stock pieces and in section 4.5 we formulate the integer linear program, which choose the limited set of patterns to pack the 2D bin. The LP relaxation of the ILP formulated in section 4.5 is solved to obtain the upper bound on the optimal value of the 2K. Finally, the section 4.6 presents the computational results of the implementation.

4.1 One-Dimensional Cutting Stock Problem

Cutting Stock Problem has an application in various industries such as steel industries, textile industries, aluminum industries etc. The 1CS problem is a well-known NP-hard problem [18], that occurs during manufacturing processes thus has gained much attention from all over the world. 1CS problem is a linear integer programming problem with one decision variable for each possible pattern. If the number of order is small, then the number of patterns may be small enough that the problem can be solved using a standard algorithm. However, there may be an exponential number of patterns if the number of order widths is large. In these cases, an alternative solution approach is needed.

In CS, given the set of items, each of which has a specific weight and a demand associated to the items, one is asked to find the minimum number of cutting patterns so that the demand is satisfied and summation of the weight of the items in each pattern does not exceed the capacity. These two problems, cutting stock and packing problem, share the same nature and the only difference between them is

- (A) 2D item with $w_i = 3$ and $h_i = 7$ (B) 1D pieces obtained after slicing the 2D item, each with $w_i = 1$ and $h_i = 7$

FIGURE 4.1: Slicing of 2D item into 1D pieces

the perspective from which the problem is investigated. More precisely, in packing problem, items are labeled; have we had several items with the same weights, BP distinguishes among them and appends a label to each item, while CS associates all the items of the same size as required demand. It obviously follows that in CS non-repeated items have demand equal to 1. Nevertheless, either formulation leads to the same optimal solution and the minimum number of the patterns (bins) is found.

4.2 One-Dimensional Cutting Stock Relaxation

In 2K problem, given the set of n rectangular items $i \in I = \{1, \dots, n\}$ each item fitting into a big rectangle (the *bin*) and is associated with parameters width w_i , height h_i and positive value p_i . The item is then packed into the bin with the edge parallel to the edges of the rectangular bin. Without loss of generality, we assume that $h_i \leq H$ and $w_i \leq W$ and all the dimensions are integers, where (W, H) is the width and height of the big rectangular bin to be packed.

Vertical Relaxation

The intuition under this relaxation is easily explained as follows. We imagine cutting the 2D items along the height into pieces of unit width, and do the same with the bin:

<i>Items</i>	(w_i, h_i)	LB_i	UB_i	π_i
1	(3,7)	0	2	35
2	(8,2)	0	2	40
3	(10,2)	0	1	27
4	(5,4)	0	3	23
5	(2,9)	0	2	43

TABLE 4.1: Set of 2D items that should be packed into the bin of size (10,10): size, minimum and maximum number of each item that is allowed to pack into the bin and their respective value.

as a result, the i -th item generates w_i 1D pieces of height h_i , and the bin generates W pieces (regarded as 1D stock pieces) of height $H \geq h_i$. We call this relaxation as *Vertical Relaxation*. The Figure 4.1 shows the slicing of 2D item into 1D pieces. We then try to obtain small pieces for a total maximum value by solving a modified 1CS problem where, unlike the classical case, only W stock items are available and item i , if cut from stock, must be produced in the demanded amount w_i (from now on, we reserve the term *item* to 2D parts, the term *piece* to 1D ones and the term *stock pieces* for the 1D stock generated after slicing the bin). The value of the items is divided uniformly into the number of pieces the item is cut.

The described relaxation does not remove inequalities from a particular formulation, but rather weakens the geometrical properties of items (viewed as real 2D intervals), that is *connectivity*. A further relaxation is done by removing integrality constraints in the 1CS problem. This removal simplifies computation, as the linear relaxation can be computed by column generation. Column generation is successfully employed to compute very good linear relaxations of CUTTING STOCK (CS).

To provide the illustration of the 1-Dimensional Cutting Stock relaxation of 2-Dimensional Knapsack Problem we define a small instance of the 2K problem. We consider the big rectangle of size (10,10) and the set of items to be packed is given in table 4.1. In the table, for any item i , (w_i, h_i) and π_i represent size and value of item i , respectively and LB_i and UB_i represent the minimum and maximum number of the item of type i that can be packed into the bin. Table 4.2 shows the size, lower and upper bound and value of the items after they are sliced into 1D pieces.

<i>Pieces</i>	(w_i, h_i)	LB_i	UB_i	V_i
1	(1,7)	0	6	11.66
2	(1,2)	0	16	5
3	(1,2)	0	10	2.7
4	(1,4)	0	15	4.6
5	(1,9)	0	4	21.5

TABLE 4.2: Set of the 1D pieces obtained after slicing 2D items.

4.3 Problem Formulation

We have obtained the 1D pieces and stock piece after slicing them from the 2D items and bin, respectively. Now, we need to determine the minimum number of stock pieces that should be used to satisfy the total demand of the pieces. The possible model for the problem is defined as follows.

Sets

- $I = \{1, 2, \dots, n\}$: set of piece types;
- J : set of patterns (i.e. possible ways) that a stock piece can be cut into pieces of required lengths.

Parameters

- H : length of the stock piece;
- h_i : length of piece i , $\forall i \in I$;
- d_i : demand for each piece i , $\forall i \in I$;
- a_{ij} : number of pieces of type $i \in I$ in pattern $j \in J$.

Decision Variable

- x_j : number of stock pieces that should be cut using pattern $j \in J$.

Model

$$\min \quad \sum_{j \in J} x_j \quad (4.1)$$

$$\text{s.t.} \quad \sum_{j \in J} a_{ij} \cdot x_j \geq d_i \quad \forall i \in I \quad (4.2)$$

$$x_j \in \mathbb{Z}^+ \quad \forall j \in J \quad (4.3)$$

4.4 Column Generation

The Integer Linear Programming model (4.1-4.3) is very elegant but it assumes the availability of parameters a_{ij} and also the set I . Hence, in order to generate this data one needs to enumerate all the cutting patterns. It is easy to realize that in real-world instances the number of possible cutting patterns is relatively very huge and hence the direct implementation of the above model is unpractical. In practical situations, the demands are so high that the number of stock pieces cut is also very large, and therefore a good heuristic solution can be obtained by rounding up to the next integer variable x_j found by solving the continuous relaxation. Hence we start with the solving the continuous relaxation of the above model.

4.4.1 Master Problem

The continuous relaxation of the model (4.1-4.3), also known as Master Problem (MP), is as follows.

$$\min \quad z_{MP} = \sum_{j \in J} x_j \quad (4.4)$$

$$\text{(MP)} \quad \text{s.t.} \quad \sum_{j \in J} a_{ij} \cdot x_j \geq d_i \quad \forall i \in I \quad (4.5)$$

$$x_j \in \mathbb{R}^+ \quad \forall j \in J \quad (4.6)$$

where x_j is the number of times pattern $j \in J$ is used, $|J| = N$ is the set of patterns, a_{ij} is the number of pieces of type i obtained from pattern j and d_i is the demand for each piece i . Figure 4.2 shows an example of a pattern where we can see that $a_{2j} = 1$, $a_{3j} = 2$, $a_{4j} = 1$, $a_{ij} = 0 \quad i \neq 2, 3, 4$.



FIGURE 4.2: Example of a pattern

4.4.2 Restricted Master Problem

We know that the number of possible patterns is obviously very large, notice that N is exponential in n because feasible patterns are in one-to-one correspondence with the solution of a complex packing problem (e.g., KNAPSACK). Therefore, the MP cannot be solved as it is, and we consider the Restricted Master Problem (RMP) which contains a limited subset J' of patterns (with the only initial condition that every item i has a non-zero entry in at least one pattern $j \in J'$):

$$\min \quad z_{RMP} = \sum_{j \in J'} x_j \quad (4.7)$$

$$(RMP) \quad \text{s.t.} \quad \sum_{j \in J'} a_{ij} x_j \geq d_i \quad \forall i \in I \quad (4.8)$$

$$x_j \geq 0, \quad j \in J' \quad (4.9)$$

Any feasible solution for the RMP can be constructed as follows:

- Consider the single-piece cutting patterns, i.e., I patterns, each containing $a_{ii} = H/h_i$ piece of type i .
- However, we expect to retain the geometry of an item after packing (i.e. while rearranging the pieces after packing we expect to obtain the original shape of each item), hence; the number of pieces of each item i that is allowed in the stock piece is bounded by UB_i , which is the maximum number of items of type i that is allowed to be packed into the bin. Hence, in the feasible cutting pattern $a_{ii} = UB_i$ can be considered as the starting point.

As shown in figure 4.3, we can see that for *pattern1* we have $a_{11} = 1$ and $a_{i1} = 0, \forall i \neq 1$, for *pattern2*, $a_{22} = 2$ and $a_{i2} = 0, \forall i \neq 2$ and similarly for the other pieces. Thus following the above procedure leads to the diagonal the constraint matrix \bar{B} .

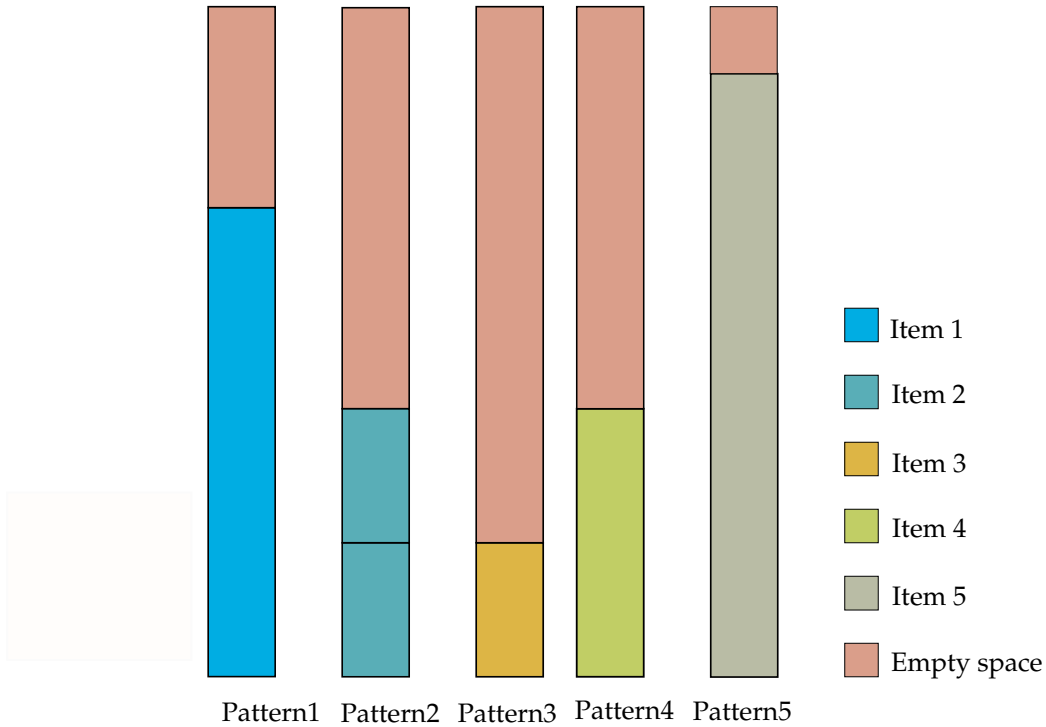


FIGURE 4.3: Initial set of patterns for solving the RMP

$$\begin{aligned}
& \min \sum_{j \in J'} x_j \\
& \text{s.t. } \bar{A}x_j \geq d_i \\
& \quad x_j \geq 0
\end{aligned}
\quad \text{where} \quad \bar{A} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 2 & 0 & \dots & 0 \\ 0 & 0 & a_3 & \dots & \vdots \\ \vdots & & & \ddots & \vdots \\ 0 & \dots & \dots & \dots & a_n \end{bmatrix}, \quad (4.10)$$

n represents the number of pieces and a_i is the number of same pieces i that can be packed into a stock piece. However, the optimal solution for the RMP need not be the optimal solution for the MP. Let π be the dual variable. Then the dual of the RMP is as follows,

$$\max \sum_{i \in I} \pi_i d_i \quad (4.11)$$

$$\text{s.t. } \sum_{i \in I} a_{ji} \pi_i \leq 1 \quad \forall j \in J' \quad (4.12)$$

$$\pi_i \geq 0, \quad i \in I \quad (4.13)$$

We solve the above RMP and its dual to optimality, thus obtaining the primal optimal solution z_{RMP}^* and the dual optimal solution π^* that satisfy the complementary slackness condition. The purpose of the RMP is to provide dual variable values: To communicate with the pricing problem which primal variables are needed to come closer to dual feasibility, and thus primal optimality.

Now, we need to generate the columns that have the negative reduced cost. However, not only the number of possible columns is exponentially large but the columns are also not known explicitly. In order to find (if any) a column (primal variable) with the negative reduced cost, it is necessary to solve a Pricing Problem (PP).

4.4.3 Pricing Problem

The pricing problem provides a column that prices out profitably or proves that none exists. In order to see whether the variable with a negative reduced cost exists it is sufficient to solve the following 1D INTEGER KNAPSACK problem, called as Pricing Problem (PP).

$$\begin{aligned}
& \min \{1 - \pi^* y\} \\
& \text{s.t.} \quad y \text{ is the feasible column of the constraint matrix}
\end{aligned} \tag{4.14}$$

We know that every column of the constraint matrix is a cutting pattern; hence, any

	<i>Piece1</i>	<i>Piece2</i>	<i>Piece3</i>	<i>Piece4</i>	<i>Piece5</i>	<i>Value</i>
<i>Pattern1</i>	1	0	1	0	0	14.36
<i>Pattern2</i>	0	2	1	1	0	17.29
<i>Pattern3</i>	0	0	1	2	0	11.89
<i>Pattern4</i>	0	1	0	2	0	21.50
<i>Pattern5</i>	0	0	0	0	1	14.19
<i>Pattern6</i>	1	1	0	0	0	16.66

TABLE 4.3: Set of patterns generated by the column generation algorithm

feasible column matrix should satisfy the condition $\sum_{i \in I} h_i y_i \leq H$. Also, in our case we want to retain the original geometry of each item. This means, in each column, the number of pieces of an item should be less than or equal to the maximum number of items of type i is allowed to pack into the bin. For every column generated we need the constraint $y_i \leq UB_i \quad \forall i \in I$ in the pricing problem. In the example, upper bound on the number of times *Item1* can be packed into the bin is 2 hence $y_1 \leq 2$. So, This leads to the following integer linear programming,

$$\min \quad 1 - \sum_{i \in I} \pi_i^* y_i \tag{4.15}$$

$$\text{s.t.} \quad \sum_{i \in I} h_i y_i \leq H \tag{4.16}$$

$$y_i \leq UB_i \quad \forall i \in I \tag{4.17}$$

$$y_i \in \mathbb{Z}^+ \tag{4.18}$$

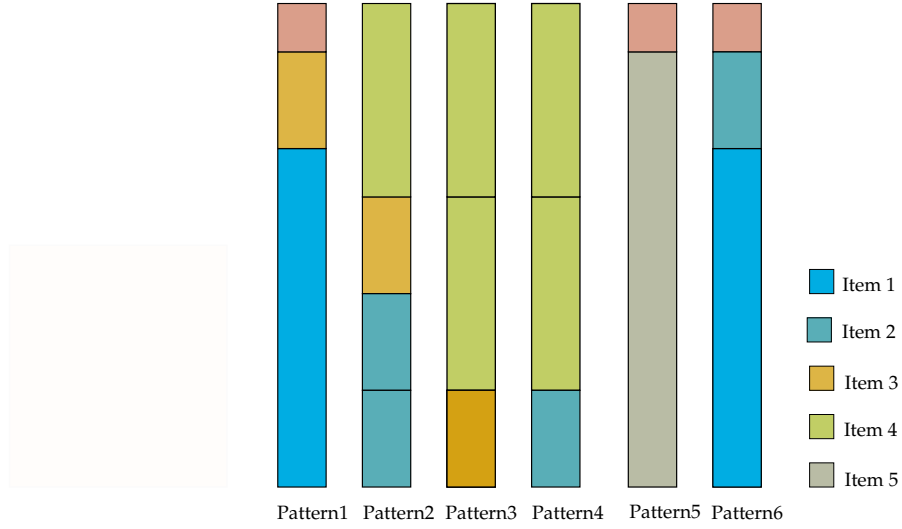


FIGURE 4.4: Patterns generated by column generation algorithm.

which is equivalent to the following,

$$\max \quad \sum_{i \in I} \pi_i^* y_i \quad (4.19)$$

$$\text{s.t.} \quad \sum_{i \in I} h_i y_i \leq H \quad (4.20)$$

$$y_i \leq UB_i \quad \forall i \in I \quad (4.21)$$

$$y_i \in \mathbb{Z}^+ \quad (4.22)$$

where h_i is the height of the piece i , H is the length of the stock piece where the pieces of items are packed and UB_i is the maximum number of item of type i which is allowed to be packed into the bin. A solution y of the PP gives a new column of the RMP and the resulting problem is solved again by a pivot step. The new dual optimum defines a new PP and the process is repeated until no column with negative reduced cost is found. Hence when the maximum and finite optimal solution to the equation (4.19-4.22) is found, we add the variable with the coefficient column to the RMP and this process is repeated until there are no longer the variables with the negative reduced costs. Value of each pattern is equal to the sum of the value of each piece present in the pattern. Recall that the demand for each item is equal to the number of pieces the item is cut. In the above example, solving the column generation leads to the generation of the patterns, as shown in table 4.3 and figure 4.4.

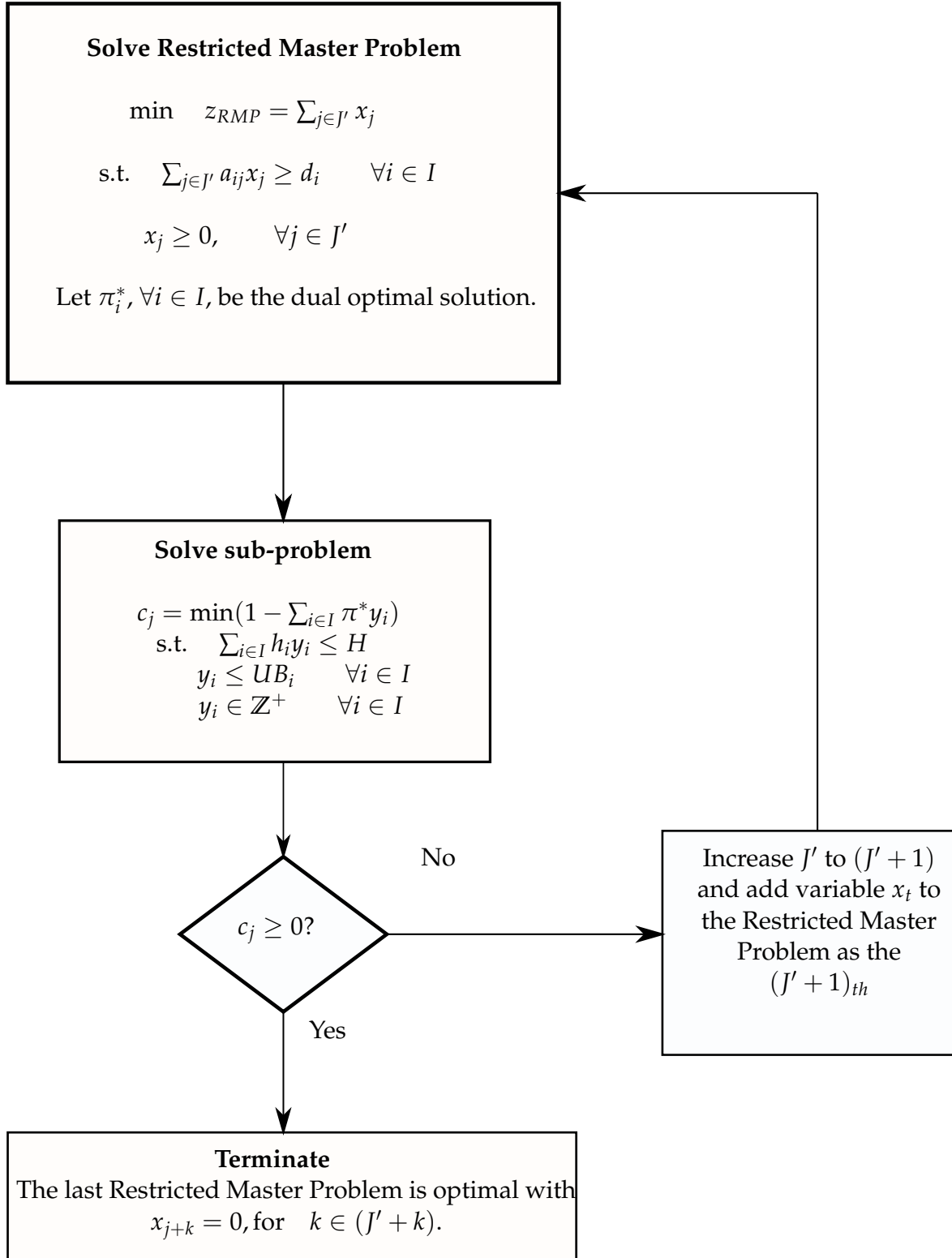


FIGURE 4.5: Flowchart: Column Generation

4.5 ILP formulation for packing patterns into bin

In our case, an upper bound to 2D INTEGER KNAPSACK is computed by choosing a limited set of W patterns so that the demand of each piece is fully satisfied or the piece is not produced at all, and the total value of the pieces produced is maximized. Recall that the value of each piece is divided uniformly into the number of the piece, each item is cut. Hence the value of each pattern is equal to the sum of value pieces present in the pattern.

Now, we formulate the following integer linear program (ILP) (4.23-4.28) to pack the 1D stock pieces, obtained after column generation, to the bin while maximizing the total value.

$$\max \quad \sum_i \pi_i y_i \quad (4.23)$$

$$\text{s.t.} \quad \sum_k p_{ik} x_k = b_i y_i \quad \forall i \in I \quad (4.24)$$

$$\sum_k x_k = W \quad (4.25)$$

$$LB_i \leq y_i \leq UB_i \quad \forall i \in I \quad (4.26)$$

$$y_i \in \mathbb{Z}^+ \quad \forall i \in I \quad (4.27)$$

$$x_k \geq 0 \quad (4.28)$$

where y_i is the number of items i produced, x_k is the run length of pattern k , p_{ik} is the number of units of piece i produced by pattern k , b_i is the demand of each piece of type i (= item width or the number of strips in which item i is sliced), π_i is the value of item i , and LB_i and UB_i is the minimum and maximum number of items of type i that can be produced, respectively. In the above program, the first set of equations i.e. equation (4.24) fulfills the demand of piece i whenever $y_i = 1$. Equation (4.25) ensures that the number of unit width patterns chosen are equal to the width of the stock rectangle and equation (4.26) provides bound on the number of items that can be packed into the bin. The LP relaxation of the integer linear program is solved to obtain the upper bound on the maximum value of the items packed into the given bin.

To continue with the illustration of the relaxation, the LP relaxation of the above ILP gives the following optimal value,

$$\begin{aligned} x_1 &= 0, x_2 = 2, x_3 = 0, x_4 = 4, x_5 = 4 \\ y_1 &= 0, y_2 = 1, y_3 = 0, y_4 = 2, y_5 = 2 \end{aligned}$$

Hence, the total maximum value of the item packed into the bin of size (10, 10) is

$$= \pi_2 \times 1 + \pi_4 \times 2 + \pi_5 \times 2 = 40 + 2 \times 23 + 2 \times 43 = 40 + 46 + 86 = 172$$

The knapsack problem is solved so that the demand of each piece is fully satisfied or the piece is not produced at all, and the total value of the pieces produced is maximized. The set of patterns chosen are shown in the Figure 4.6. Now, re-arranging the set of patterns leads us to the final packing, as shown in the Figure 4.6d. In this case, we can see that the *Item5* and *Item4* have the original shape of the item, while the *Item2* is selected just because the demand for it is satisfied i.e. the width of *Item2* which is 8 pieces. Hence this provides us the upper bound to the optimal solution. However, if the solution achieved is optimal then, in that case, we can retain the geometry of every item in the bin.

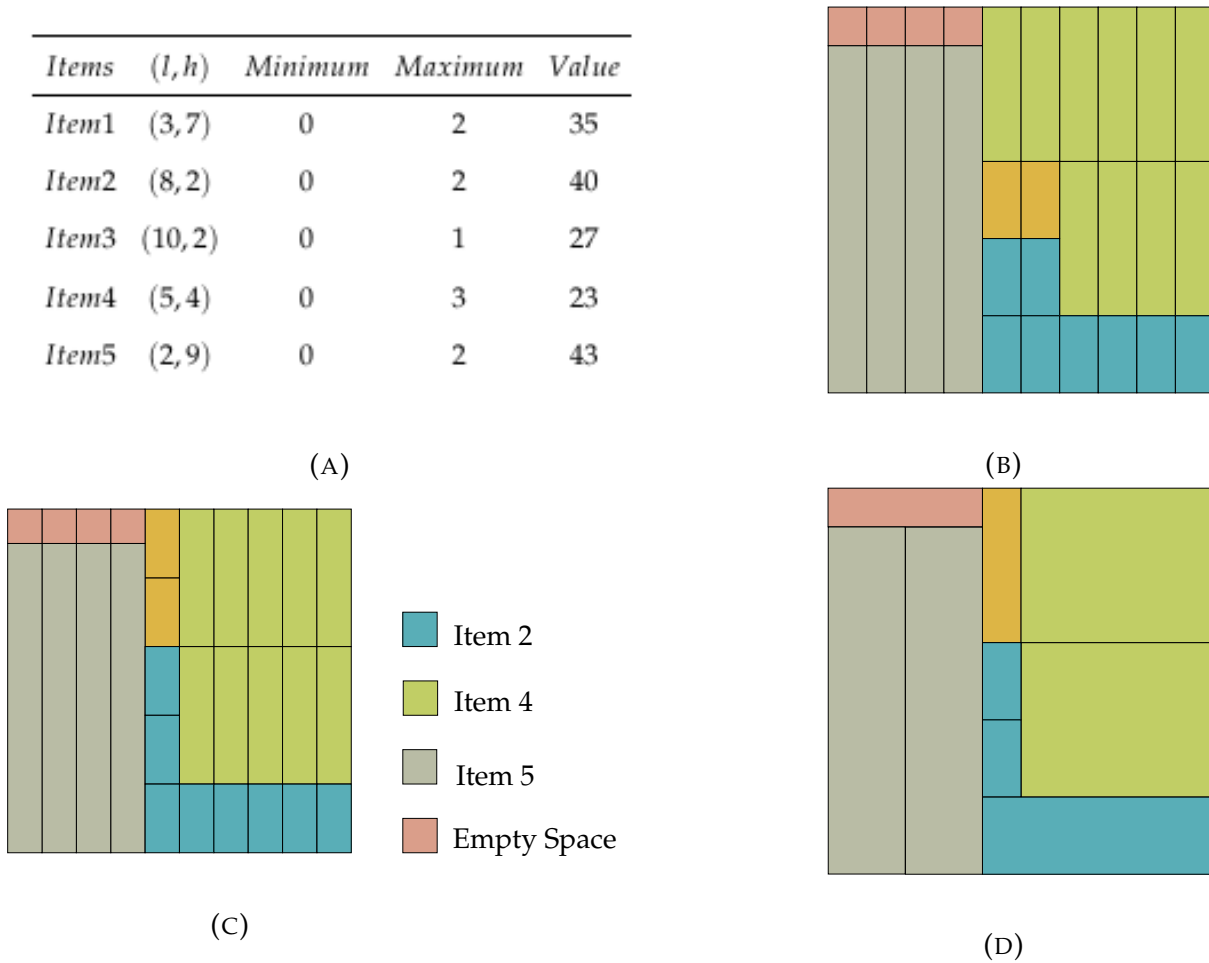


FIGURE 4.6: Stages of 2D packing by 1D relaxation

Horizontal Relaxation

Similarly, we performed the relaxation by slicing the items along the width i.e. by replacing W, w_i by H, h_i in the above relaxation. We call this relaxation as *Horizontal Relaxation*. The upper bound obtained are represented as UB_{ver} and UB_{hor} , respectively. We choose the best bound as the minimum of the two.

4.6 Computational Results

For computational implementation, we use the benchmark data sets provided in Beasley [5]. The instances are divided into four classes.

- Class1: $W = H = 10$ and w_i and h_i are random in $[1, 10]$.
- Class2: $W = 15, H = 10$ and w_i and h_i are random in $[1, 15]$.
- Class3: $W = H = 20$ and w_i and h_i are random in $[1, 20]$.
- Class4: $W = H = 30$ and w_i and h_i are random in $[1, 30]$.

where W and H are the width and height of the bin and w_i and h_i are the width and height of item i , respectively. Each class of data set has 3 instances with n items ($n = 5, 7, 10$).

The implementation was made in Intel Core i5-3210M with 8GB of RAM. The algorithm was programmed in Python and solved using GUROBI optimizer. The computed results are presented in Table 4.6. Every problem instance is relaxed vertically and horizontally and the best of the two is chosen for comparison.

In the table, UB and OPT are upper bound and optimal solution from [5]. UB_{ver} and UB_{hor} represent the upper bound obtained with vertical and horizontal relaxation as described in Chapter 4, respectively. UB_{best} is the best upper bound i.e. $\min(UB_{ver}, UB_{hor})$, $Gap\ vs\ UB = (UB_{best} - UB)$ and $Gap\ vs\ OPT = (UB_{best} - OPT)$.

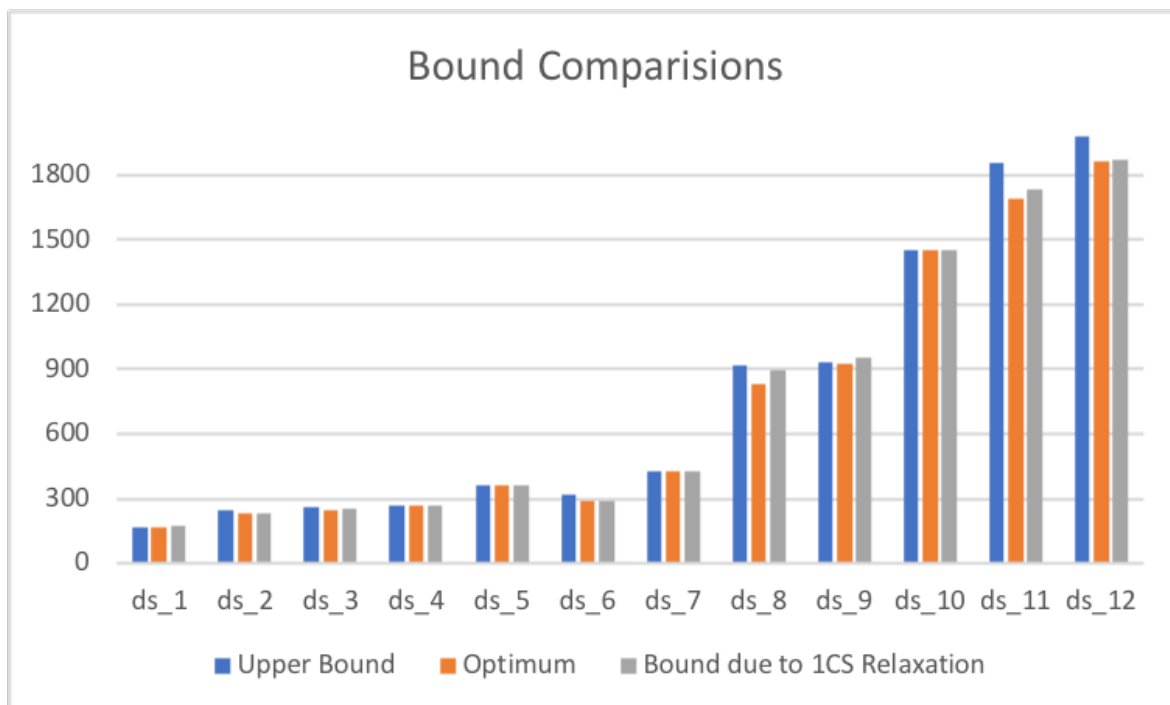


FIGURE 4.7: Comparison of Bounds obtained due to 1CS relaxation with the optimal value.

Problem No.	Bin Size	Number of Items	UB	OPT	UB _{ver}	Time (Sec)	UB _{hor}	Time(Sec)	UB _{best}	Gap vs UB	Gap vs OPT	%(Gap vs OPT)
1	(10,10)	5	164	164	172	0.05	201	0.03	172	8	8	4.88
2	(10,10)	7	247	230	250	0.04	232	0.06	232	-15	2	0.87
3	(10,10)	10	260	247	254	0.04	253	0.04	253	-7	6	2.43
4	(15,10)	5	268	268	268	0.03	268	0.03	268	0	0	0
5	(15,10)	7	358	358	373	0.04	358	0.05	358	0	0	0
6	(15,10)	10	317	289	317	0.03	291	0.07	291	-26	2	0.69
7	(20,20)	5	430	430	430	0.01	430	0.03	430	0	0	0
8	(20,20)	7	915	834	897	0.05	938	0.03	897	-18	63	7.55
9	(20,20)	10	930	924	955	0.05	953	0.05	953	23	29	3.14
10	(30,30)	5	1452	1452	1452	0.03	1517	0.05	1452	0	0	0
11	(30,30)	7	1860	1688	1737	0.05	1864	0.04	1737	-123	49	2.91
12	(30,30)	10	1982	1865	1875	0.09	1983	0.06	1875	-107	10	0.54

TABLE 4.4: Computation Results

Chapter 5

Conclusions

Our aim in this thesis was to investigate the upper bounds to the optimal solution (which is the maximum value of the items packed) of the 2K problem by 1CS relaxation. In doing so, we applied the DW decomposition and column generation algorithm to obtain the 1D relaxation of the 2D packing problem. Our overall aspiration was to study the upper bounds to the optimal solution obtained due to relaxation. This work can be divided into three key categories: presenting the relevant mathematical background, problem-specific formulation, and the computational implementation.

- **Theoretical Background**

We have presented the theoretical background of the decomposition principle and column generation which was crucial for initiating the work. The decomposition principle is an important framework in dealing with the problem which becomes intractable as the size of the problem increases and the column generation algorithm is a technique to implement the decomposition. We also introduced the notion of Dual Feasible Function (DFF) which can be a good starting point for the future research.

- **Problem-Specific Formulation**

We introduced the problem-specific formulation for generating columns, which is the addition of the knapsack constraint in the pricing problem. This constraint took care of the feasible relaxation i.e retaining the original geometry of an item. Afterward, an integer linear program was formulated to pack the 1D pieces obtained after column generation. We solved the LP relaxation of the ILP to obtain upper bound to the optimal solution.

- **Computation**

The proposed methodology on the relaxation was implemented and solved efficiently using PYTHON and GUROBI. In order to provide the proof-of-concept

of the functionality of the relaxation methodology, we considered twelve benchmark datasets whose optimal value were already known and the upper bounds were suggested. The empirical results reported from all the experiments based on the relaxation showed the possible profitability and benefit of the proposed approach.

Future Research

We conclude our thesis by providing the direction for the future research, which follows the research done in this thesis. The possible directions can be on strengthening the bound and fastening the bound computation.

- **Branch and Price**

When the 1CS-like formulation (whose solution yields the upper bound) has exponentially many columns, optimal integer solutions can be found by branch-and-price. The performance of a B&P procedure inherently depends on the quality of the upper and lower bounds found at each node, and more importantly, relies on the use of computer resources (branching strategies, search strategies, column management etc.) [17]. In general terms, B&P uses bounds from LP relaxations solved by column generation at branching nodes. While doing B&P, a non-basic column of the Restricted Master Problem (RMP) that appears in a branch is not necessarily non-basic in other branches. So, a well-thought-out problem-specific heuristic column management that makes use of knowledge from the original problem would arguably improve the algorithm performance. Another important aspect where the performance of B&P can be improved is the branching scheme. Intuitively, the B&P search tree is heuristically constructed via node-picking and branching rules. The node-picking rule decides where in the tree further branching or pruning should be done, and a bad rule can give rise to time-consuming branches. Branching rules have been investigated by Achterberg et al.[2], and hybrid ones by Achterberg and Berthold [1]. Problem-specific branching and node-picking rules can be investigated and developed in the present context where we do not refer to a standard 1CS, but rather to a 1CS-like problem. Also, the use of computer resources can have a tremendous effect on algorithm performance. Nowadays, almost every processor is available with multiple cores. Hence, it is natural to thread up independent subproblems to multiple cores, so as to solve them in parallel. A more sophisticated approach would solve multiple B&P nodes

in parallel, with a caveat that, although parallel computing of independent subproblems is intuitive, parallelization of B&P is quite involved: so, how to distribute local information to parallel processes can be the interesting direction of research.

- **DFF**

Another interesting area of research can deal with the use of DFF to fasten bound computation. DFF [3] provide dual feasible solutions of models whose corresponding bounds are often very close to those provided by column generation. As one can design a DFF to be computed quickly, the computational burden can be very small compared to that obtained by column generation. Furthermore, for a given problem, it is often possible to derive not only a single DFF but several DFF, or even families of different DFF, providing possibly different bounds from which taking the best one.

Bibliography

- [1] Tobias Achterberg and Timo Berthold. Hybrid branching. In Willem-Jan van Hoeve and John N. Hooker, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 309–311, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
- [2] Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. *Oper. Res. Lett.*, 33(1):42–54, January 2005.
- [3] Cláudio Alves, François Clautiaux, José Carvalho, and Jürgen Rietz. *Dual-Feasible Functions for Integer Programming and Combinatorial Optimization: Basics, Extensions and Applications*. 02 2016.
- [4] Niclas Andreasson, Anton Evgrafov, Michael Patriksson, Emil Gustavsson, and Magnus nnheim. *Introduction to Continuous Optimization*. Studentlitteratur, 2 edition, 2013.
- [5] J. E. Beasley. An exact two-dimensional non-guillotine cutting tree search procedure. *Operations Research*, 33(1):49–64, 1985.
- [6] Gleb Belov, Vadim Kartak, Heide MeiSSner, and Guntram Scheithauer. Onedimensional relaxations and lp bounds for orthogonal packing. 16:745 – 766, 11 2009.
- [7] Andreas Bortfeldt and Tobias Winter. A genetic algorithm for the two-dimensional knapsack problem with rectangular pieces. 16(6):685–713.
- [8] R C E Gilmore and Ralph Gomory. A linear programming approach to the cutting stock problem i. 9, 01 1961.
- [9] Alberto Caprara and Michele Monaci. On the two-dimensional knapsack problem. 32:5–14, 01 2004.
- [10] Nicos Christofides and Eleni Hadjiconstantinou. An exact algorithm for orthogonal 2-d cutting problems using guillotine cuts. *European Journal of Operational Research*, 83(1):21 – 38, 1995.

- [11] Nicos Christofides and Charles Whitlock. An algorithm for two-dimensional cutting problems. 25:30–44, 02 1977.
- [12] George B. Dantzig and Philip Wolfe. Decomposition Principle for Linear Programs. *Operations Research*, 8(1):101–111, February 1960.
- [13] Jacques Desrosiers and Marco E Lübbecke. A primer in column generation. In *Column generation*, pages 1–32. Springer, 2005.
- [14] Sándor P. Fekete, Jörg Schepers, and Jan van der Veen. An exact algorithm for higher-dimensional orthogonal packing. *CoRR*, abs/cs/0604045, 2006.
- [15] Sandor P. Fekete and Jörg Schepers. On more-dimensional packing i: Modeling, 2000.
- [16] Aleksei Fishkin, Olga Gerber, Klaus Jansen, and Roberto Solis-Oba. Packing weighted rectangles into a square, 08 2005.
- [17] DAVID FRIBERG. An implementation of the branch-and-price algorithm applied to opportunistic maintenance planning.
- [18] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.
- [19] P. C. Gilmore and R. E. Gomory. Multistage cutting stock problems of two and more dimensions. *Operations Research*, 13(1):94–120, 1965.
- [20] P.C. Gilmore and Ralph Gomory. A linear programming approach to the cutting stock problem part ii. 11, 12 1963.
- [21] Iwama K. Han, X. and G. Zhang. Theory comput syst (2008) 43: 38.
- [22] J. C. Herz. Recursive computational procedure for two-dimensional stock cutting. *IBM J. Res. Dev.*, 16(5):462–469, September 1972.
- [23] Klaus Jansen and Guochuan Zhang. Maximizing the total profit of rectangles packed into a rectangle. 47:323–342, 03 2007.
- [24] Erwin Kalvelagen. Column generation with gams. 2003.
- [25] Ariel Kulik and Hadas Shachnai. There is no EPTAS for two-dimensional knapsack. 110(16):707–710.

- [26] Jr. L. R. Ford and D. R. Fulkerson. A suggested computation for maximal multi-commodity network flows. *Management Science*, 50(12_supplement):1778–1780, 2004.
- [27] Katherine Lai. The knapsack problem and fully polynomial time approximation schemes (FPTAS). page 6.
- [28] Marco E. Lübbecke and Jacques Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.
- [29] JoséFernando Oliveira and JoséSoeiro Ferreira. An improved version of wang’s algorithm for two-dimensional cutting problems. *European Journal of Operational Research*, 44(2):256 – 266, 1990. Cutting and Packing.
- [30] Guntram Scheithauer. Introduction to cutting and packing optimization. 263, 01 2018.
- [31] François Vanderbeck and Laurence Wolsey. An exact algorithm for ip column generation. 19:151–159, 10 1996.
- [32] P Y. Wang. Wang, p.y.: Two algorithms for constrained two-dimensional cutting stock problems. *operations research* 31(3), 573–586. 31:573–586, 06 1983.