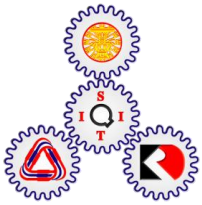


HEURISTICS FOR TWO-DIMENSIONAL RECTANGULAR GUILLOTINE CUTTING

BY

KIMSENG TIENG

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF ENGINEERING (LOGISTICS AND SUPPLY CHAIN SYSTEMS ENGINEERING)

SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY

THAMMASAT UNIVERSITY

ACADEMIC YEAR 2015

HEURISTICS FOR TWO-DIMENSIONAL RECTANGULAR GUILLOTINE CUTTING

BY

KIMSENG TIENG

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF ENGINEERING (LOGISTICS AND SUPPLY CHAIN SYSTEMS ENGINEERING)

SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY

THAMMASAT UNIVERSITY

ACADEMIC YEAR 2015

HEURISTICS FOR TWO-DIMENSIONAL RECTANGULAR GUILLOTINE CUTTING

A Thesis Presented

By

KIMSENG TIENG

Submitted to

Sirindhorn International Institute of Technology

Thammasat University

In partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING (LOGISTICS AND SUPPLY CHAIN SYSTEMS ENGINEERING)

Approved as to style and content by

Advisor and Chairperson of Thesis Committee _____

(Assoc. Prof. Dr. Chawalit Jeenanunta, Ph.D)

Committee Member and
Chairperson of Examination Committee _____

(Asst. Prof. Dr. Aussadavut Dumrongsiri, Ph.D)

Committee Member _____

(Assoc. Prof. Dr. Ruengsak Kawtummachai, Ph.D)

DECEMBER 2015

Abstract

HEURISTICS FOR TWO-DIMENSIONAL RECTANGULAR GUILLOTINE CUTTING

by

KIMSENG TIENG

Bachelor of Science, Major Physics,
Royal University of Phnom Penh, 2013

Two-Dimensional Rectangular Guillotine Cutting (2DRGC) is an important problem in the industrial. Paper, sheet, plate, glass, wood, or plastic is needed to cut from stock in big rectangular area. Then a set of small sheet, plate, glass, wood, or plastic in rectangular size is given. Guillotine is a cutting process, where the sheets are needed to cut straightly. In this thesis, eight different methods are implemented to find a good layout. They are 2D horizontal construction, 2D vertical construction, 2D horizontal improvement, 2D vertical improvement, Sheet Width Panel Height Horizontal Cut, Sheet Width Panel Width Vertical Cut, Minimum Sheet Width Ordering Panel Height Horizontal Cut, and Minimum Sheet Height Ordering Panel Width Vertical Cut. Given result of each method is used to compare with 2D simple heuristic cutting and column generation method. The objective of this thesis is to find the best layout within a short computational time, and demand of customer is fulfilled. The result indicates that a good layout has been found in a short computational time within some proposed heuristics.

Keywords: Heuristics, 2D guillotine cutting stock, Column generation,

Acknowledgements

This thesis will not accomplish if I did not get support and help from my advisor and committees, Sirindhorn International Institute of Technology (SIIT), faculty members, and colleagues, and especially my family and relatives.

First of all, I would like to reveal my deeply thankfulness to my advisor, Dr. Chawalit Jeenanunta, for his invaluable guidances, motivation, and immense knowledge throughout my limited time frame research at SIIT, Thailand. He is my best advisor ever. Besides my advisor, I would like to express my sincere gratitude to my internal and external committee Dr. Aussadavut Dumrongsiri and Dr. Ruengsak Kawtummachai for every useful comments and suggestions through the entire process of my research. Those guidances are extremely vital to push my research moving forward; otherwise, I cannot graduate in time.

Secondly, I am highly indebted to SIIT, Thammasat University for their constant supervision and continuous support by providing me an Excellent Foreign Scholarship (EFS) scholarship. It is a golden opportunity to pursue my master degree. In addition, many thanks and appreciation also go to all faculty members, staffs and my colleagues in Logistics and Supply Chain System Engineering program (LSCSE) for their helpfulness.

Last but not least, I would like to express my special gratitude and thankfulness to both my parents and my siblings, as well as my relatives. They always support, love, encourage, and stay by my side in all kind of situation.

Table of Contents

List of Tables

List of Figures

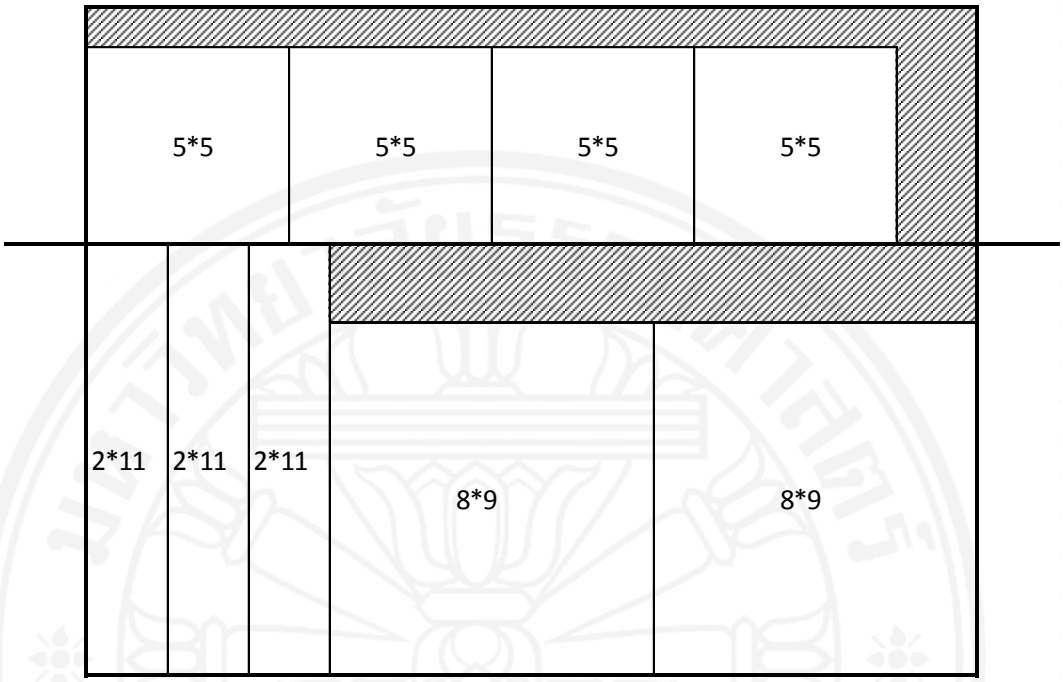
Chapter 1
Introduction

1.1. Background

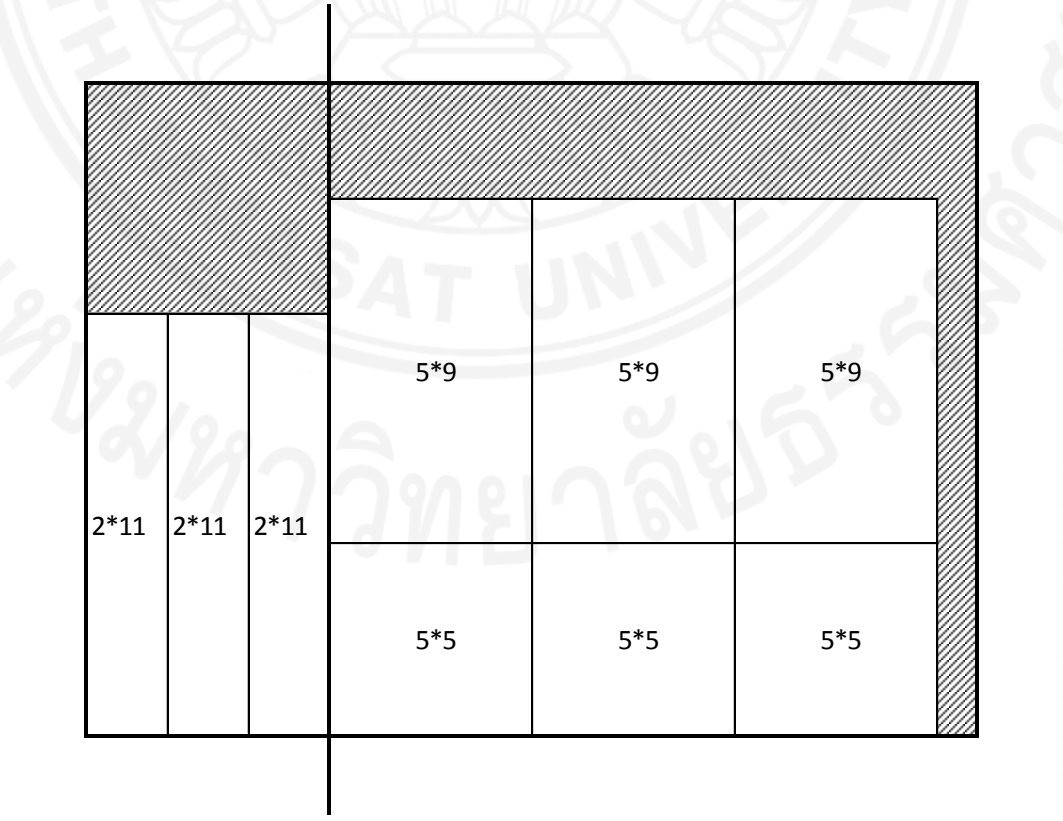
Two-Dimensional Rectangular Guillotine Cutting (2DRGC) is an important problem in the cutting industrial. Paper, sheet, plate, glass, wood, or plastic is needed to cut from a stock in big rectangular area such that a set of small sheet, plate, glass, wood, or plastic in rectangular size is given. In this thesis, eight different methods are implemented to find a good layout. Many scholars proposed different methods in this field namely: linear programming, dynamic programming, integer linear programming, integer programming, column generation, heuristics, and meta-heuristic method. Among these methods, heuristics become a popular method. Heuristics cannot give an optimal layout as the exact algorithm, but it can provide a good layout in a short computational time no matter what size of problem is.

Three different styles of cutting, 2D 2stage 1group, 2D 2stage 2group, and t-shape 3group, are mentioned in Figure 1.1. Guillotine is a cutting process where each sheet is needed to cut straightly. Group is the number of categories after sheet cutting. Cutting each group can produce many strips. Strip can store only one level of panel inside it. Panel is a result of strip cutting. Total number of times, raw materials are rotated, called stage.

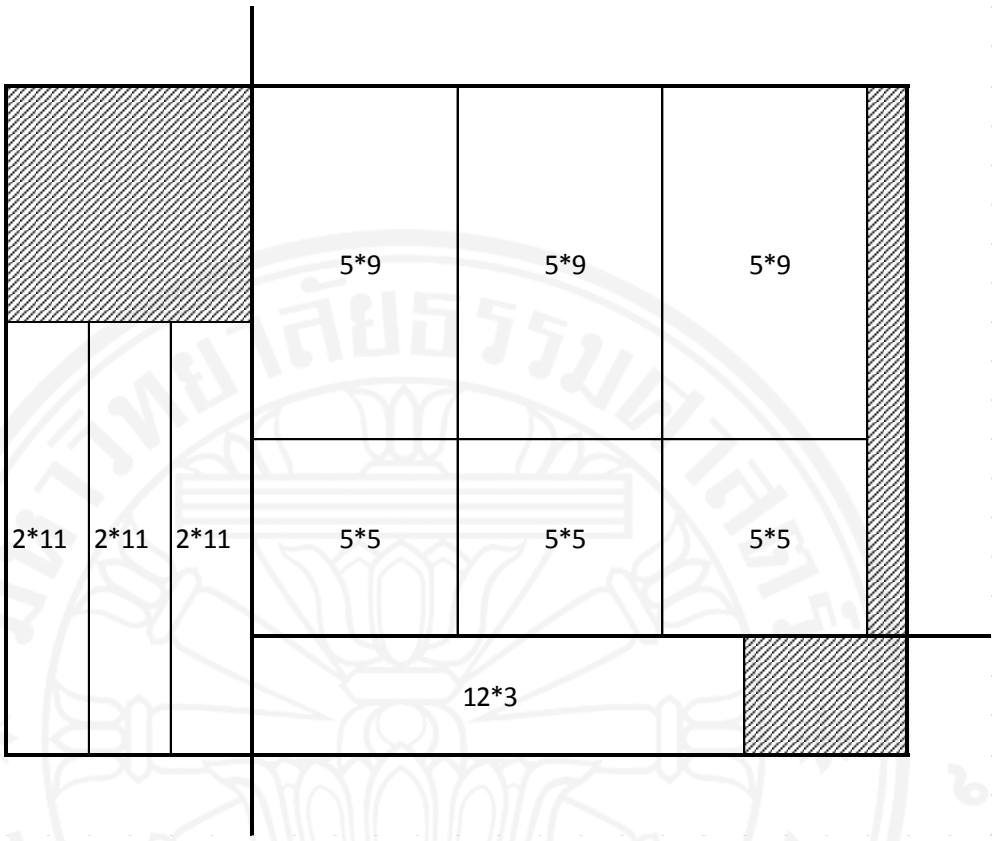
Figure 1.1(A), displays 2D 2stage 1group cutting. A horizontal cutting is applied for the first stage to produce the strips. Then each strip is rotated to cut to produce panel in the second stage. Another step of cutting is needed to separate waste and panel, but it is not counted as stage. Figure 1.1(B) displays 2D 2stage 2group cutting. Two sub-groups are given as a result of sheet cutting. Then each sub-groups is applied 2D 2stage 1group. Figure 1.1(C) displays t-shape 3group cutting. A vertical cut is applied to the sheet to produce two sub-groups, and one of the sub-groups is needed to cut to produce another two sub-groups. In total, three sub-groups are produced. Then each sub-groups is cut, using 2D 2stage 1group cutting.



A.



B.



C.

Figure 1.1 Different types of guillotine cutting methods

1.2. Problem Description

A small instance is selected to use in this thesis. Given a set of panel $P = \{p_1, p_2, ..., p_j\}$ coordinate with panel width set $w = \{w_1, w_2, ..., w_j\}$, panel height set $h = \{h_1, h_2, ..., h_j\}$, and demand of each panel type set $d = \{d_1, d_2, ..., d_j\}$ where $j = 1,2,3,...,n$. The demand each panel type is needed to cut from a set of sheet $S = \{s_1, s_2, ..., s_i\}$ coordinate with sheet width set $W = \{W_1, W_2, ..., W_i\}$, and sheet height set $H = \{H_1, H_2, ..., H_i\}$, where $i = 1,2,...,m$. In this case, the capacity of each sheets type in stock is unlimited. Rotation of the sheet is prohibited. Demand for each panel type must be fulfilled. Only guillotine cut is used. The over cut panels are regarded as waste.

1.3. Thesis Objective

The objective of this thesis is to find the best layout by making sure that the demand of customer is fulfilled. A good layout have been found in a short computational time within some heuristic methods.

1.4. Significant of the Thesis

This research is based on an Electronics Board Cutting Public Company Limited, where the electronic boards are ordered by the customers in different sizes and amounts. Replying to this order quantities, the company has to cut the rectangular sheet in big size into the panels in small rectangular size from the raw material, existed in stock. Every single year this company spent millions of dollars on the waste. The price per unit of the produced products is expansive. Then the profit for each unit of products in the company is also low. This problem is caused by the implementation of a simple way of cutting. They place only one size of panels into one size of sheet. This method cannot provide a minimum wastes for each cutting pattern.

Applying a heuristic cutting method is the best choice for this company since the good patterns of cutting can be given in a short computational time for large size of instances. It can combine different panel type into each size of sheet to cut. Thus, it is very essential to improve cutting pattern such that raw material utilization is maximized, and a competitive price for each unit of products can be sold by the company to the customers.

1.5. Overview of Thesis

The structure of this thesis is organized by doing the literature review in section 2. All proposed methods are mentioned in section 3. These methods are tested with twenty different sizes of instances in section 4. Then section 5 mentions the experimental result and discussion of each method. The conclusion from the experiment is drawn in section 6. Finally the appendices including code for each heuristic method, input data for all instances, and the output structure for each method is given in the appendices part.

Chapter 2
Literature Review

Cutting stock is a widely problem in the operation research for both one-dimensional cutting and two-dimensional cutting. It has been extensively treated in different literatures by different researchers. They uses different methods to find the best patterns to cut the sheet, cloth, steel, paper, glass, furniture, textiles, and metallurgy with different criteria. Those methods are mentioned as in the following.

2.1. Linear Programming

Gilmore and Gomory (1965) proposed linear programming (LP) model, using knapsack problem to deal with 1D and 2D cutting stock problem. In their paper, a wide class of cutting stock problems have been restricted.

Lodi and Monaci (2003) has proposed an integer linear programming models for 2D 2stage knapsack problems. The performance of their paper is analyzed in term of the quality in accordance with the classical column generation approach. They use standard branch-and-bound algorithm, and implemented it in CPLEX 6.5.3. The objective of their research is to maximize the total number of panels used and the total profit of cut panels.

H. H. Yanasse and Morabito (2008) have proposed a note on integer linear programming models to generate pattern in 2-group and 3-group constrained and unconstrained 2D guillotine cutting stock. This model is extended from linear models for 1group guillotine cut. They select a randomly generated and an actual instances to measure the performance of the model. Implemented this model in GAMS modelling and CPLEX solver, they find that this model is not efficient for large size problem.

2.2. Mix Integer Programming

Andrade, Birgin, and Morabito (2013) have proposed a mix integer programming to deal with non-exact 2D 2stage guillotine cutting problems with usable

leftover. If the plate remainder of the cutting pattern or trim loss is large enough, it can be reused for the next cutting. It is characterized as a residual bin-packing problem since the possibility of producing new residual panels did not regarded as waste. As long as the trim loss size is large enough to cut to produce the panels, it can be considered as sheet. An objective of their paper is to minimize the cost among all possible solution. It only choose the patterns that the value of generated usable leftover is maximized.

2.3. Dynamic Programming

Yaodong Cui and Liu (2007) have proposed rectangular T-shape homogenous block patterns for 2D cutting problem by using a dynamic programming recursion to generate optimal blocks. Two segments of sheets are a result of a vertical cut. Each segment consists of sections that have the same length and direction. Each section stores a row of homogenous blocks. A homogenous block consists of homogenous strips of the same panel type. The computational result indicates that the algorithm is efficient in improving material utilization, and the computational time.

Yaodong Cui (2012c) has proposed a new dynamic programming procedure for three-staged cutting patterns to solve 2D cutting stock problem. He presents unconstrained three-staged patterns algorithm. The objective of his study is to maximize the pattern value. This method can provide an efficient solution in short computational time.

2.4. Meta-Heuristics

There are different kinds of method in the meta-heuristics namely: Simulated Annealing (SA), Ant System (AS), Bee Algorithms (BA), Particle Swarm Optimization (PSO), Genetic Algorithms (GAs), Tabu Search (TS), and Harmony Search Algorithm (HSA).

SA is a random stochastic optimization method proposed by Kirkpatrick and Vecchi (1983). It was given by $T(t) = T0\alpha t$ where t was the counter of iterations, T0 was the initial temperature, and α varied from 0 and 1, and T(t) was the temperature

at iteration t. This problem appropriated for a continuous problem in the energy of physical system.

AS was a stochastic search method. It is used to solve both discrete and continuous optimization problem. It was proposed by Drigo, Maniezzo, and Colorni (1996) to optimize by using a colony of cooperating agents. This method mimicked the behavior of real ants to solve NP-hard combinatorial optimization problem. AS used exploration and exploitation to search and improve the solution. Even this method generated only a limited number of different solution when they applied greedy algorithms. It was typically better than a completely random generated solution. The final solution of AS was also depend on the initial solution.

BA was population-based combinatorial search algorithm developed by Pham et al. (2006). This method mimicked the food foraging behaviors of swarms of honey bees to look for the best solution. It could be applied to solve discrete and continuous problem. First, the initialization of the algorithm was needed. Then it searched for the solution iteration by iteration. The given solution was evaluated by comparing the fitness function. Each iteration composed of five main steps. They were recruitment, local search, neighborhood shrinking, site abandonment, and global search. This process was terminated when the number of iteration was met or the solution was acceptable.

PSO was a random stochastic continuous nonlinear optimization function proposed by Kennedy and Eberhart (1995). Two main components in PSO were; firstly, connecting to artificial life in general. Secondly, connecting to fish schooling, bird flocking, and swarm theory in specifically. This method had a simple concept. It could be implemented with a few line of code without affected to the computational time and the memory requirement. From this paper, every individual in the school could get the benefits from the discoveries, then they shared the information among their school to improve those benefits.

GAs was a stochastic combinatorial natural search optimization method proposed by Holland (1975). This method revealed about the genetic operator. It adapted the environment such that it could be used to search and evaluate the solution based on the fitness function in an effective and efficient way. Three operators, used to develop new members of the population, were reproduction, crossover, and mutation.

GAs could search many peaks, and it could reduce the possibility of local minimum trapping.

TS was a stochastic discrete combinatorial optimization methods proposed by Glover (1977). It was used to explore the feasible solution by using the sequence of move from one solution to another.

HSA is one of a recent stochastic meta-heuristics search method. It mimics the musical process. This method is applicable in discrete and continuous variables. It is a population based searching method. It has been proposed by Zong Woo Geem, Kim, and Loganathan (2001). No papers, applied HSA to solve 2D rectangular guillotine cutting problem, is found. Hence, the main objective of this research paper is to apply a HSA to deal with a static optimization in the 2DRGC.

Ayachi.I, Kammarti.R, Ksouri.M, and Borne.P (2010) have applied HSA to deal with a container storage problem to determine the best containers arrangement such that it can meet customers delivery dates and reduce the number of container movement. In this paper, they consider the influences of the number of containers, stopping criteria value, and the harmony memory size as the fitness function. The result indicates that the value of the fitness function of the last iteration is lower than the first iteration. The higher the value of the stopping criteria is the better of the quality of the fitness function. The higher of the HMS is the better of the value of the fitness function. Comparing this method with GAs, they find that HSA can give a better value of fitness function in all size of the containers.

Zong Woo Geem (2008) has summarized all the problems, using HSA. They are vehicle routing Zong W Geem, Lee, and Park (2005), and Pichpibul and Kawtummachai (2013), discrete structural optimization Lee, Geem, Lee, and Bae (2005), water distribution network Zong Woo Geem (2006), multiple dam scheduling, dome truss design, grillage structure design, grillage structure design, mix proportioning of steel and concrete, satellite heat pipe design, petroleum structure mooring, fluid transport minimal spanning tree, parameter calibration of flood routing model, parameter calibration of rainfall-runoff model, energy-saving pump operation, pipeline of oil well heat waste, soil slope stability, large-scale irrigation network design, and web-based optimization.

Worasucheep (2011) have proposed an improved HS, called harmony search with adaptive pitch adjustment (HSAPA) for continuous optimization problem. This paper has found that the higher of HMCR can provide a better performance in general, and the performance of HSAPA is shown not to be sensitive to its new parameters.

2.5. Heuristics

Yaodong Cui (2012a) has proposed a CAM system 1D cutting stock problem. M types of panels are cut from stock bars of multiple sizes. The solution, given by this algorithms, is a set of cutting patterns with specified frequency. This paper generates cutting plan such that the bar cost is minimized for the primary objective. Pattern reduction and shorter stocks reduction are considered in the secondary objective. The result indicates that the algorithms can provide a better solution, compared to other pattern reduction algorithms.

Cerqueira and Yanasse (2009) proposed a pattern reduction procedure in 1D cutting stock problem by grouping items in accordance with their demands. They used a heuristic method. Cutting solution consisted of determining a group of patterns and their frequencies. Production cost relied on the changing of the pattern since it required a set up cost. The objective of this paper was to reduce the amount of different pattern in a given solution by grouping the panels in accordance with the customer demand for each panel type.

Yaodong Cui (2012b) has proposed a fast heuristics for constrained homogenous T-shape cutting patterns. He uses a heuristic method based on a dynamic programming and branch-and-bound to generate the constrained of homogenous T-shape pattern. For the first phase, plate is required to cut into homogenous strips, and each strip is divided into pieces in the second phase. The objective is to maximize the pattern value. From the computational result, he finds that the solution close to optimal, and computational time is very fast. A good initial solution can improve time efficiency lots.

Y. Cui (2004) has proposed an optimal T-shape cutting pattern for rectangular blanks. He presents an algorithm to generate guillotine cutting patterns for rectangular blanks. He tries to tradeoff between the material utilization and the

complexity of the cutting process. He allows to place only one size of panel to appear in each strip. The algorithm uses a knapsack algorithm and an implicit enumeration method to determine an optimal combination. An efficient material usage with simple cutting pattern is given in a short computational time for the presented algorithm.

Yaodong Cui and Huang (2012) have proposed a heuristics for Constrained T-shape cutting patterns of rectangular pieces. The tradeoff between the complexity of cutting and plate cost is taken into consideration. The objective of this paper is to maximize the frequency of each panel type and the pattern value. Many patterns are produced, but only maximum pattern value is selected to be a solution. The result indicates that the algorithm can provide a better material utilization solution in a fast computational time, compared to those of the optimal 2stage patterns.

HORACIO H. Yanasse, Zinober, and HARRIS (1991) have proposed a 2D cutting stock with multiple stock sizes. They use a heuristic algorithm to build the pattern. An objective of this paper is to find the best mix of boards with a minimum waste. Guillotine cut is considered to meet with an exact demand of each customer.

Suliman (2006) has proposed a sequential heuristic procedure for the 2D cutting stock problem. He presents a 3stage sequential heuristic procedure for the 2D rectangular guillotine cutting stock problem. A width cutting pattern is determined in the first stage to produces the minimum width trim loss. Determination of table length and the associated layout of the panel length mention in the second stage. Finally, a number of iterations is used to end the cutting process. An overall objective of this paper is to minimize the trim loss.

Alvarez-Valdes, Parajon, and Tamarit (2002) have proposed a computational study of LP-based heuristic algorithms for 2D guillotine cutting stock problems. They follow Gilmore and Gomory (1965) column generation scheme. For each iteration, a new cutting layout is given from the sub model. Besides a dynamic programming in the sub model, three heuristic procedures are developed to increase the complexity. Those procedures are based on GRASP and Tabu Search techniques. The computational result, given by a randomly generated test, shows that the solution is effective and efficient in term of both total waste and computational time.

Horacio Hideki Yanasse and Limeira (2006) have proposed a hybrid heuristics to reduce the number of different patterns in cutting stock problem. They

generate patterns with a limited waste. The demands are needed to fulfill when the patterns are cut. Pattern reduction techniques are applied by starting with the generated solution. The objective of this paper was to tradeoff between the total waste and the number of patterns. The result indicates that the proposed scheme gave an alternative solution to the pattern reduction problem.

Yaodong Cui, Yang, Zhao, Tang, and Yin (2013) have proposed a sequential grouping heuristics to solve 2D cutting stock with pattern reduction. They consider both input minimization as a primary objective and pattern reduction as the secondary objective. A sequential heuristic procedure is generated for each next pattern, using a dynamic programming recursion. This process is repeated until all the demand of each panel type is fulfilled. The experimental result indicates that a sequential grouping heuristics is powerful in pattern reduction, input minimization, and reduction of the computational time.

Yaodong Cui and Zhao (2013) has proposed a heuristics for the rectangular 2D single stock size cutting stock problem with 2stage patterns. A heuristic algorithm is presented to solve 2D 2stage single stock size cutting. The rotation of the panel is allowed. A column generation method is used to solve the residual problems repeatedly until customer demand for each panel type is satisfied. The computational result indicates that the algorithm can solve most instances to optimality. It is more efficient in reducing the number of plates, comparing to a published algorithm and a commercial stock cutting software package.

Chapter 3
Proposed Algorithms

In this section, various techniques are presented namely: column generation (CG), 2D simple heuristic cutting (2DSHC), 2D horizontal construction (2DHC), 2D vertical construction (2DVC), 2D horizontal improvement (2DHI), 2D vertical improvement (2DVI), Sheet Width Panel Height Horizontal Cut (SW_PH_HC), Sheet Width Panel Width Vertical Cut (SW_PW_VC), Minimum Sheet Width Ordering Panel Height Horizontal Cut (MinSW_OPH_HC), and Minimum Sheet Height Ordering Panel Width Vertical Cut (MinSH_OPW_VC).

3.1. Column Generation (CG)

Column generation technique is used to solve a combinatorial problem with many decisions. This technique bases on Danzig-Wolfe decomposition. Let $S'_i (i = 1,...,o)$ is the family of all feasible cutting pattern of sheet i. The decision variable $x_p (p \in S'_i)$ denotes the number of times when the cutting pattern p is used in the solution. A_p defines the waste from cutting pattern p. In constraints (2), the coefficients C_p^j represents the number of panel $j (j = 1,...,n)$ in the cutting pattern p, and d_j represents the requirement of panel j.

$$\textit{Minimize} \quad \sum_{i=1}^{o} \sum_{s'_i \in S'_i} A_{s'_i} x_{s'_i} \tag{1}$$

subject to

$$\sum_{i=1}^{o} \sum_{s'_i \in S'_i} C_{s'_i}^j x_{s'_i} = d_j \qquad ; j = 1,...,n \tag{2}$$

$$x_{s'_i} \in Z^+ \qquad ; i = 1,...,o \ , s'_i \in S'_i \tag{3}$$

The above model is called master problem (MP). The objective function (1) is to minimize waste of used sheets. Constraints (2) ensures that the number of panel j must equal to the requirement, and constraints (3) is the integrality constraints.

In fact, we cannot generate all feasible patterns for a large size instances. A restricted master problem (RMP) consists of a subset of patterns of master problem.

To obtain the optimal solution, we generated a sub-problem for each sheet i. We use dual solution from the current solution in RMP. The sub-problem, deal with two-dimensional two-stage knapsack problems with guillotine cuts (2DKP), is proposed by Lodi and Monaci (2003). Therefore, solving this problem can give a pattern with the most negative reduced cost to prove optimality. We show the model, which involves integer variables x_{jk} denoting the number of panels of type $j(j=1,...,n)$ in shelf $k(k=1,...,\alpha_j)$ and $q_k(k=1,...,\bar{n})$ denoting whether a shelf k is used (where $\bar{n}$ is the number of panels and $\alpha_j = \sum_{s' \le j} d_s$).

Let π_j^* be a dual solution from the current optimal solution in RMP associated with panel j. The mathematical model for each sheet type is shown below:

$$\textit{Maximize} \quad \sum_{j=1}^{n} \pi^*_j \left(\sum_{k=1}^{\alpha_j} x_{jk} + \sum_{k=\alpha_{j-1}+1}^{\alpha_j} q_k \right) \tag{4}$$

subject to:

$$\sum_{k=1}^{\alpha_j} x_{jk} + \sum_{k=\alpha_{j-1}+1}^{\alpha_j} q_k \le ub_j \qquad ; j = 1,...,n \tag{5}$$

$$\sum_{j=\beta_k}^{n} \bar{w}_j x_{jk} \le (W - \bar{w}_{\beta_k}) q_k \qquad ; k = 1,...,\bar{n} \tag{6}$$

$$\sum_{k=1}^{\bar{n}} \bar{l}_{\beta_k} q_k \le L \tag{7}$$

$$\sum_{s=k}^{\alpha_j} x_{js} \le ub_j - (k - \alpha_{j-1}) \qquad ; j = 1,...,n; k \in [\alpha_{j-1}+1, \alpha_j] \tag{8}$$

$$0 \le x_{jk} \le d_j \qquad ; x_{jk} \in \text{integer}; j = 1,...,n; k \in [1, \alpha_j] \tag{9}$$

$$q_k \in \{0,1\} \qquad ; k = 1,...,\bar{n} \tag{10}$$

The objective function (4) is to maximize the sum of the cost of panel in pattern. Inequalities (5), (6), and (7) represent the cardinality constraints, the width constraints, and the height constraint, respectively. Inequalities (8) is to strengthen the bound on the x_{jk} variables (given by inequalities (9)). If the feasible pattern with maximum profit, greater than zero, is found, the column corresponding to this pattern is added to the current RMP. When no more feasible pattern with maximum profit,

greater than zero, is found, that mean, the current optimal solution of RMP is an optimal solution of MP.

3.2. 2D Simple Heuristic Cutting (2DSHC)

2DSHC has been applied by most cutting companies. In this stage, each sheet with minimum waste in a simple heuristics for 2D cutting method contain only one panel type. The process of 2DSHC is implemented as follows:

Step 1: Given a set of sheets $S = \{s_1, s_2, ..., s_i\}$ and a set of panels $P = \{p_1, p_2, ..., p_j\}$.

Step 2: For each $p_j \in P$, select $s_i \in S$ that has a minimum waste to cut, where p_j is put in sheet s_i

Step 3: For each $p_j \in P$, number of the sheet need to cut equal to the ceiling of the ratio between the demands and the number of panels used within the sheet

Step 4: Remove $p_j \in P$ that has been cut.

Step 5: If $P = \{0\}$, terminate cutting process; otherwise, go to step 2.

For instance, given a set of panel $P = \{p_1, p_2, p_3, p_4\}$, coordinate with $w = \{2,8,5,4\}$, $h = \{11,9,5,4\}$, and $d = \{14,10,20,26\}$. These panel is cut from a sheet set $S = \{s_1, s_2\}$ coordinate with $W = \{14,22\}$ and $H = \{30,17\}$. Each panel type selects only the sheet that can give a minimum waste. The number of sheet, needed to cut for each panel types in each pattern as illustrated in Figure 3.1, are 1, 4, 2, and 2 respectively. The total waste after fulfill all the demands is 1652 in^2


2*11
2*11
2*11
2*11
2*11
2*11
2*11
8
9
10
11
12
13
14
2*11
2*11
2*11
2*11
2*11
2*11
2*11
1
2
3
4
5
6
7
8*9
3
8*9
2
8*9
1
5*5
5*5
5*5
5*5
9
10
11
12
5*5
5*5
5*5
5*5
5
6
7
8
5*5
5*5
5*5
5*5
1
2
3
4

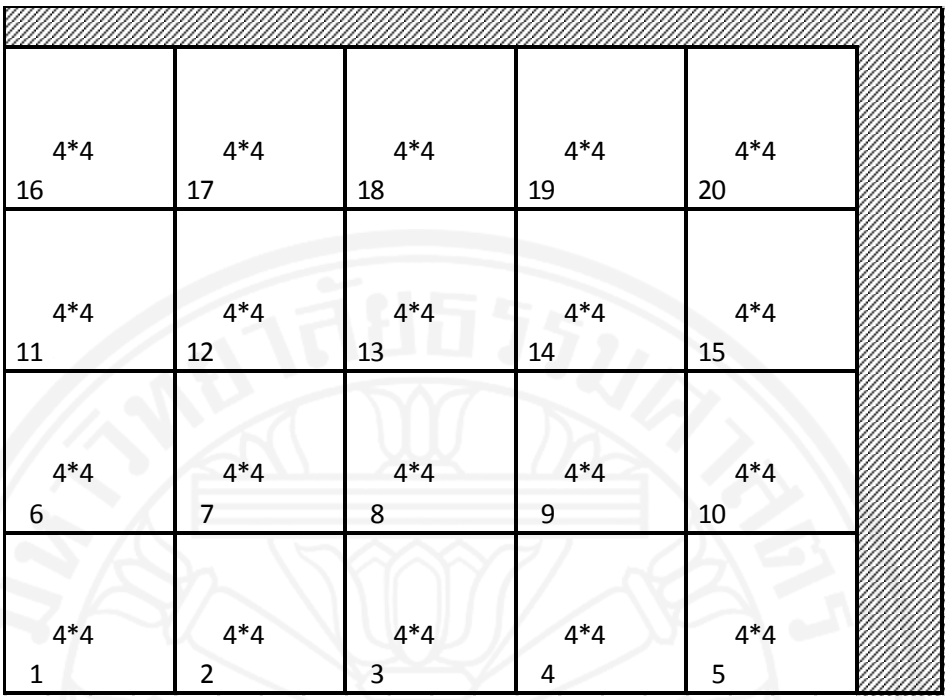


Figure 3.1 Solution of 2DSHC

= panel size (a×b) put into the sheet in of N^{th}

 = the waste of the sheet

3.3. 2D Horizontal Construction (2DHC)

In this section, the panels are arranged and cut horizontally. Multiple types of panels are allowed to put in each size of sheet. The process of this method is implemented as in the following.

Step 1: Given a set of sheets $S = \{s_1, s_2, ..., s_i\}$ and a set of panels $P = \{p_1, p_2, ..., p_j\}$. Then sort these two sets in descending order based on height.

Step 2: P' is the set of panel where $p'_j \in P'$, such that the number of p'_j with the same width and height is equal to the demand d_j of that p_j with the same width and height. Thus, $|P'| = \sum_{j=1}^{n} d_j$

Step 3: Each $p'_j \in P'$ is selected to put into the space next to the previous panel on the same strip $s_i \in S$ in horizontal line until no more panels types can be inserted in that strip, then move to next strip to cut horizontally. If there are not any strips available, move to the new sheet.

Step 4: Remove $p'_j \in P'$ that has been used.

Step 5: If $P' = \{0\}$, terminate the cutting process; otherwise, go to step 3

From the same instance mentioned in section 3.2, each panel P' is picked to cut in the sheet in the chronological order; hence, the number of sheet, needs to cut for each patterns as illustrated in Figure 3.2, are 1, 3, 1, and 1 respectively. The total waste after fulfill all the demands is 576 in^2

Figure 3.2 Solution of 2DHC

3.4. 2D Vertical Construction (2DVC)

In this section, the panels are arranged and cut vertically. Multiple types of panels are allowed to put in each size of sheet. The process of this method is implemented as in the following.

Step 1: Given a set of sheets $S = \{s_1, s_2, ..., s_i\}$ and a set of panels $P = \{p_1, p_2, ..., p_j\}$. Then sort the panel set based on width and sheet set based height in descending order.

Step 2: P' is the set of panel where $p'_j \in P'$, such that the number of p'_j with the same width and height is equal to the demand d_j of that p_j with the same width and height. Thus, $|P'| = \sum_{j=1}^{n} d_j$

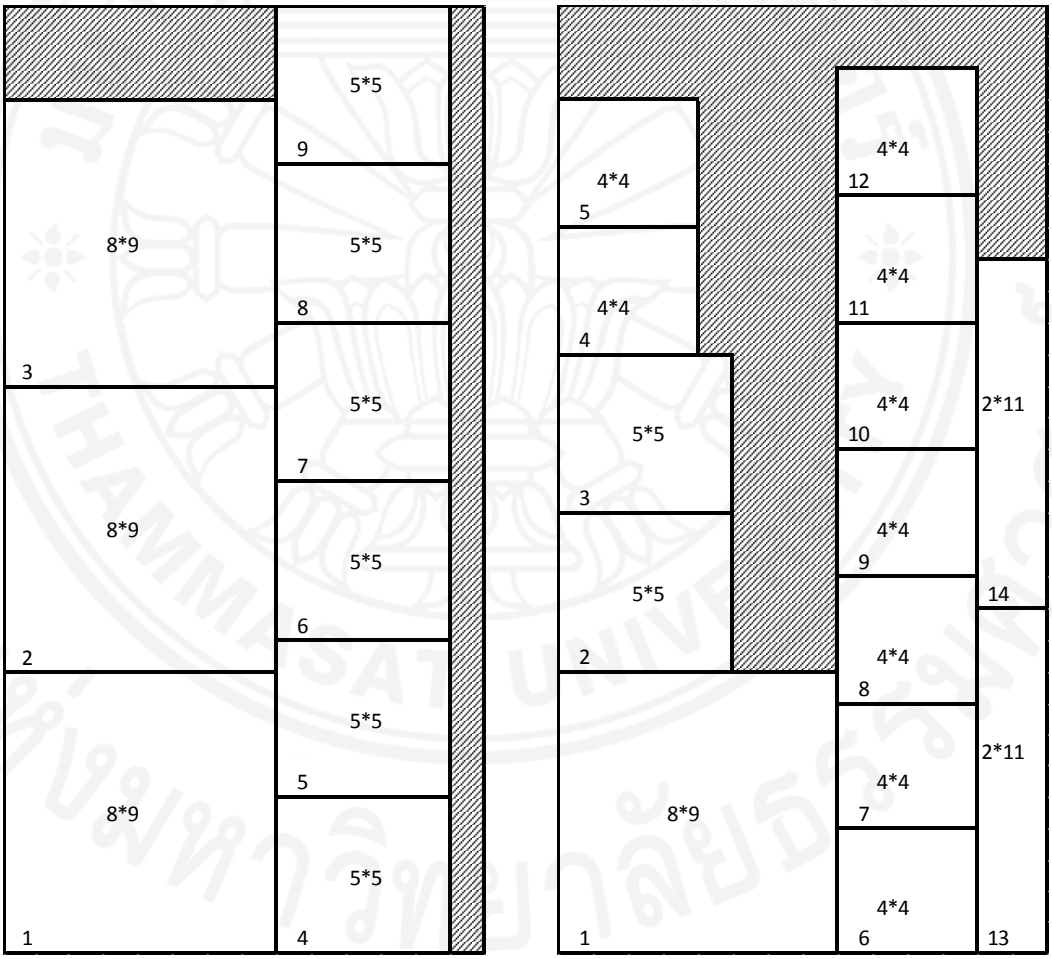
Step 3: Each $p'_j \in P'$ is selected to put into the space next to the previous panel on the same strip $s_i \in S$ in horizontal line until no more panels types can be inserted in that

strip, then move to next strip to cut horizontally. If there are not any strips available, move to the new sheet.

Step 4: Remove $p'_j \in P'$ that has been used.

Step 5: If $P' = \{0\}$, terminate the cutting process; otherwise, go to step 3

From the same instance mentioned in section 3.2, each panel P' is picked to cut in the sheet in the chronological order; hence, the number of sheet, needs to cut for each patterns as illustrated in Figure 3.3, are 3, 1, 1, and 1 respectively. The total waste after fulfill all the demands is 576 in^2



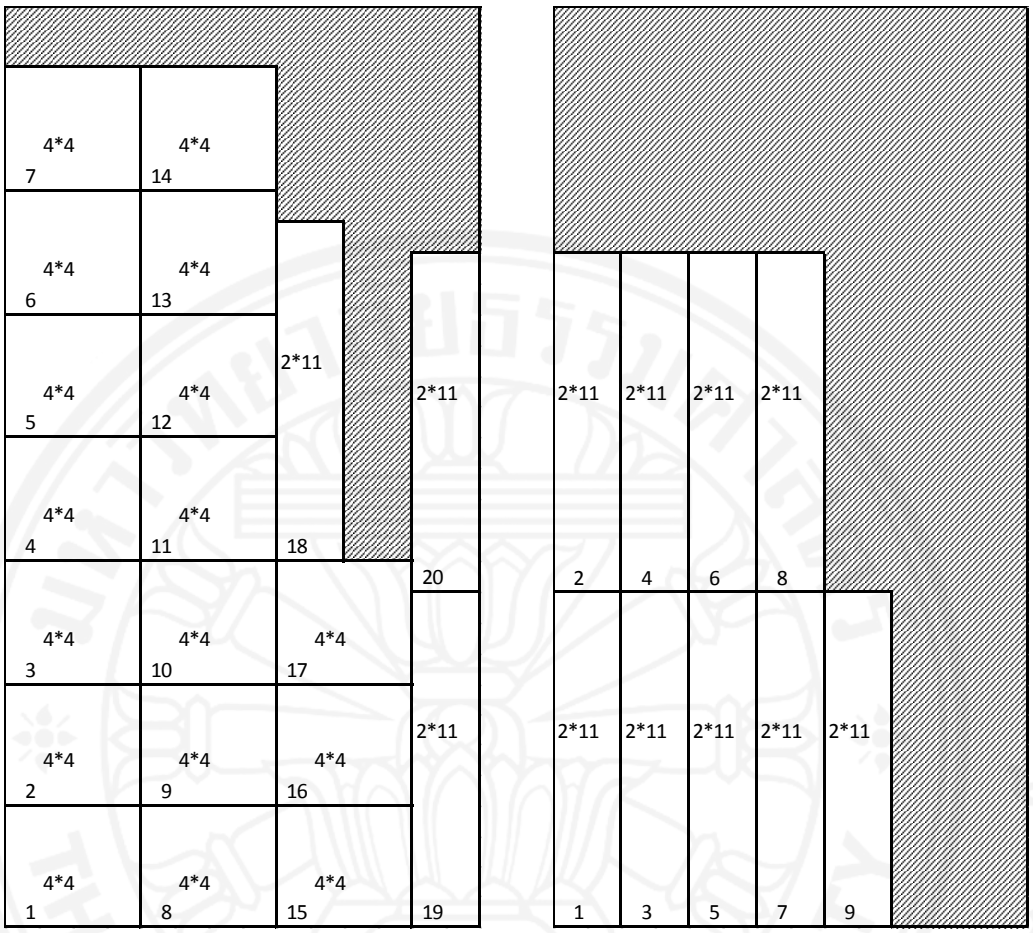


Figure 3.3 Solution of 2DVC

3.5. 2D horizontal improvement (2DHI)

In this section, the panels are arranged and cut horizontally. Multiple types of panels are allowed to put in each size of sheet. Each type of panels, locates in the first panel set, is selected to put into each type of sheets first until no more panel in that size can fill up. Then the leftover area is tried to insert the other panel types. The process of this technique is mentioned as in the following.

Step 1: Given a set of sheets $S = \{s_1, s_2, ..., s_i\}$ and a set of panels $P = \{p_1, p_2, ..., p_j\}$. Then sort the panel set in descending order based on height. P' is also the set of panel where $p'_j \in P'$, such that the number of p'_j with the same width and height is equal to the demand d_j with the same width and height. Thus, $|P'| = \sum_{j=1}^{n} d_j$.

Step 2: For each $p'_j \in P'$, put p'_j in each sheet s_i as much as possible in horizontal line. Once the horizontal strip is filled up, move to the next strip vertically until there is no more space to put p'_j

Step 3: For each $s_i \in S$ used in step 2, select the next $p'_j \in P'$ to put into the leftover area. Put p'_j in sheet s_i as much as possible in horizontal line. Once the horizontal strip is filled up, move to the next strip vertically until there is no space to put p'_j. Repeat this step for the next p'_j

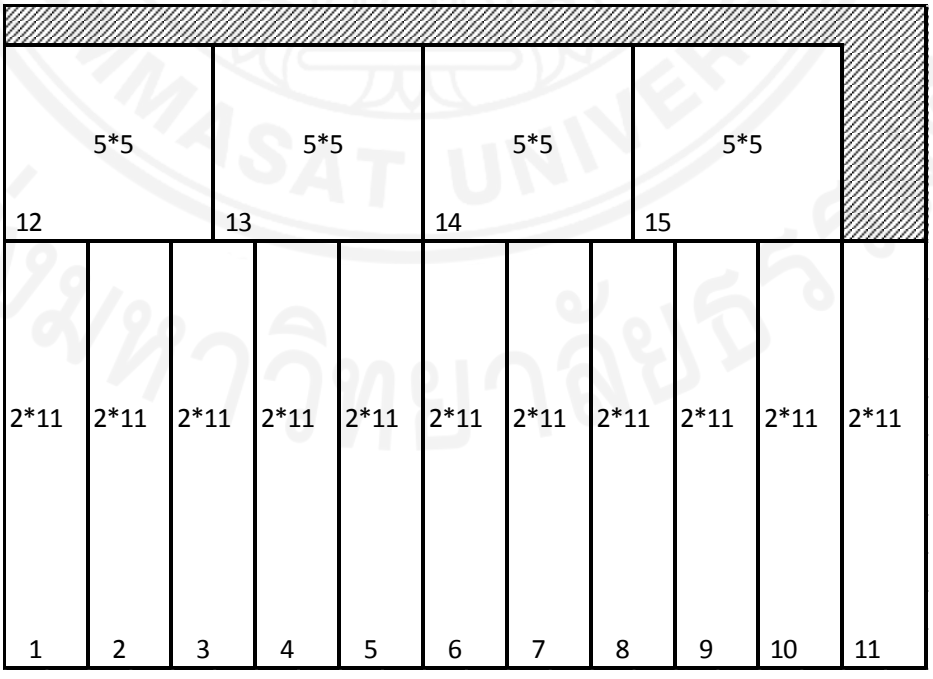
Step 4: Select $s_i \in S$ are used in step 3 that can give a minimum waste to cut.

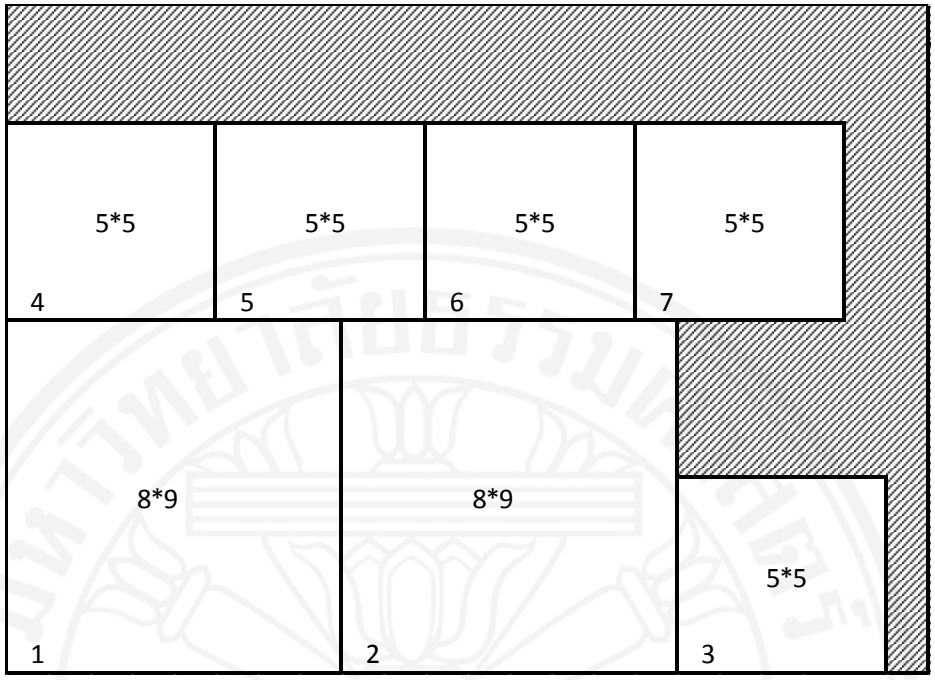
Step 5: Number of the sheet need to cut is equal to the ceiling of the smallest ratio between the demands and the number of each panels types used in the pattern

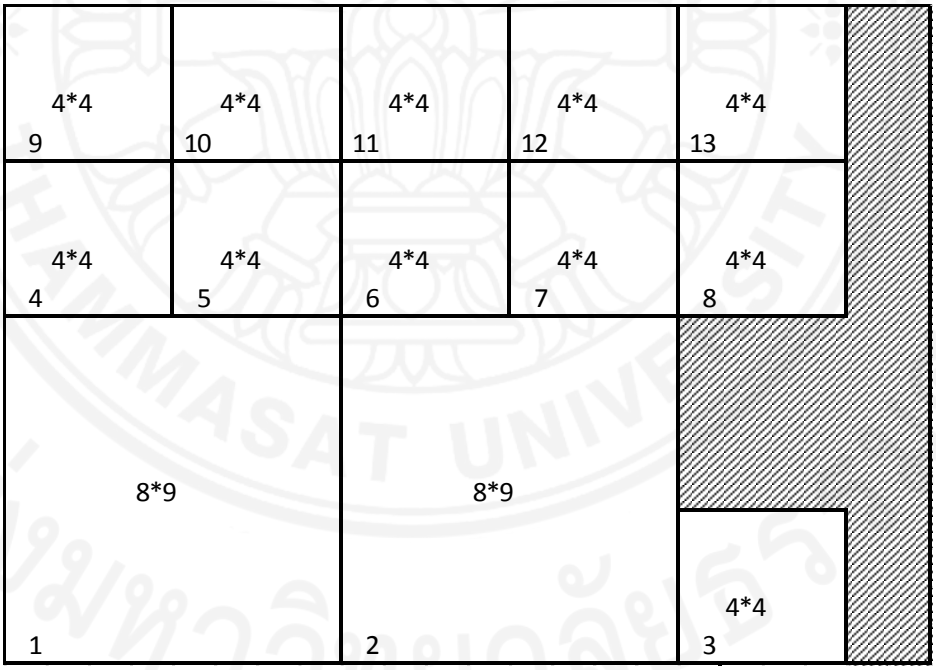
Step 6: Remove $p'_j \in P'$ that has been used.

Step 7: If $P' = \{0\}$, terminate the cutting process; otherwise, go to step 3

From the same instance mentioned in section 3.2, each panel P' is picked to cut into the sheet in the chronological order; hence, the number of sheet, needs to cut for each patterns as illustrated in Figure 3.4, are 2, 3, 2, and 1 respectively. The total waste after fulfill all the demands is 1048 in^2




5*5
4
5*5
5
5*5
6
5*5
7
8*9
1
8*9
2
5*5
3


4*4
9
4*4
10
4*4
11
4*4
12
4*4
13
4*4
4
4*4
5
4*4
6
4*4
7
4*4
8
8*9
1
8*9
2
4*4
3

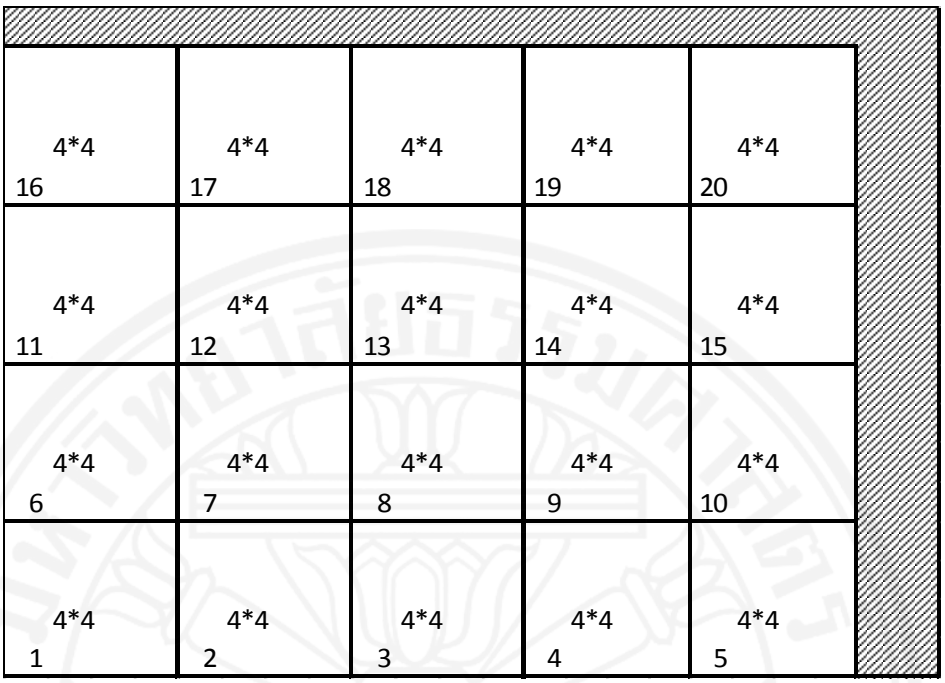


Figure 3.4 Solution of 2DHI

3.6. 2D Vertical Improvement (2DVI)

In this section, the panels are arranged and cut vertically. Multiple types of panels are allowed to put in each size of sheet. Each type of panels located in the first panel set is selected to put into each type of sheets first until no more panel in that size can fill in. Then the leftover area is tried to insert the other panel types. The process of this technique is mentioned as in the following.

Step 1: Given a set of sheets $S = \{s_1, s_2, ..., s_i\}$ and a set of panels $P = \{p_1, p_2, ..., p_j\}$. Then sort the panel set in descending order based on width. P' is also the set of panel where $p'_j \in P'$, such that the number of p'_j with the same width and height is equal to the demand d_j with the same width and height. Thus, $|P'| = \sum_{j=1}^{n} d_j$.

Step 2: For each $p'_j \in P'$, put p'_j in each sheet s_i as much as possible in vertical line. Once the vertical strip is filled up, move to the next strip horizontally until there is no more space to put p'_j

Step 3: For each $s_i \in S$ used in step 2, select the next $p'_j \in P'$ to put into the leftover area. Put p'_j in sheet s_i as much as possible in vertical line. Once the vertical strip is filled up, move to the next strip horizontally until there is no space to put p'_j. Repeat this step for the next p'_j

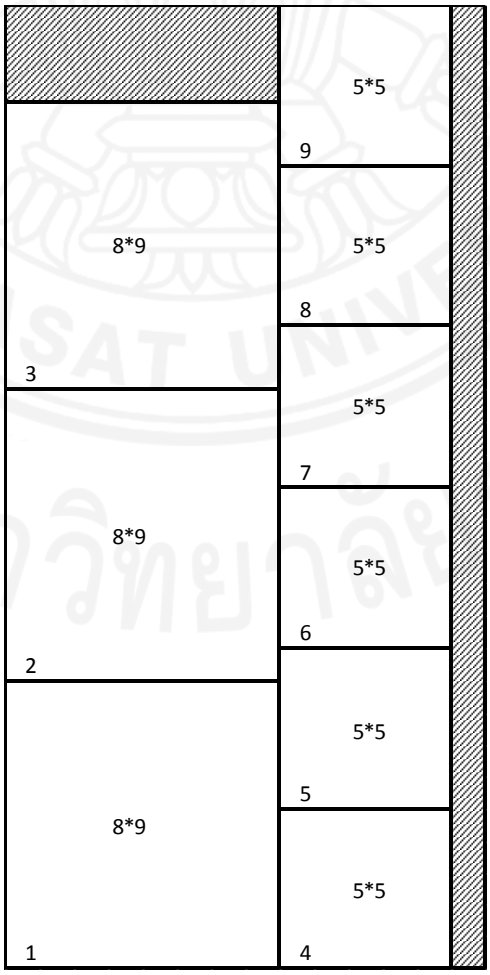
Step 4: Select $s_i \in S$ are used in step 3 that can give a minimum waste to cut.

Step 5: Number of the sheet need to cut is equal to the ceiling of the smallest ratio between the demands and the number of each panels types used in the pattern

Step 6: Remove $p'_j \in P'$ that has been used.

Step 7: If $P' = \{0\}$, terminate the cutting process; otherwise, go to step 3

From the same instance mentioned in section 3.2, each panel P' is picked to cut in the sheet in the chronological order; hence, the number of sheet, needs to cut for each patterns as illustrated in Figure 3.5, are 4, 2, and 1 respectively. The total waste after fulfill all the demands is 904 in^2.



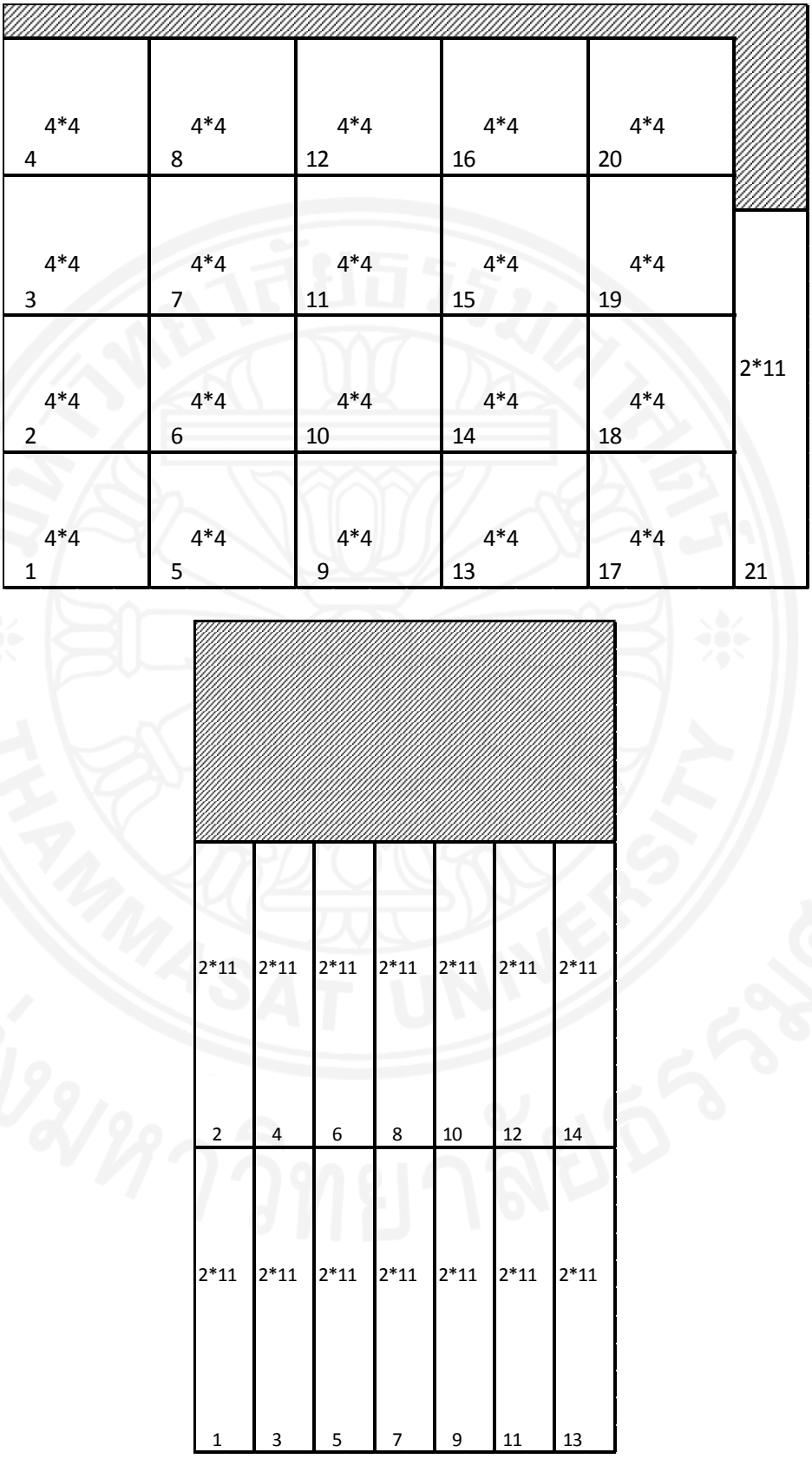


Figure 3.5 Solution of 2DVI

3.7. Sheet Width Panel Height Horizontal Cut (SW_PH_HC)

In this section, the panels are arranged to cut horizontally. Multiple types of panels are allowed to put into each size of the sheet. The process of this method is implemented as in the following.

Step 1: Given a set of sheets $S = \{s_1, s_2, ..., s_i\}$ and a set of panels $P = \{p_1, p_2, ..., p_j\}$. Sort the sheet based on width, and sort the panel based on height in descending order.

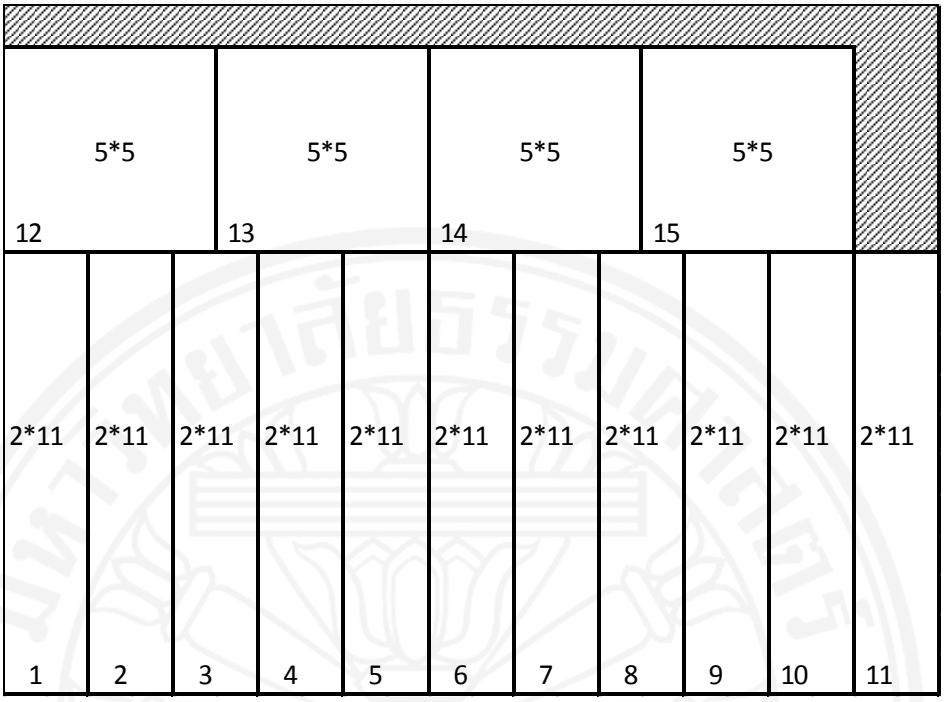
Step 2: P' is the set of panel where $p'_j \in P'$, such that the number of p'_j with the same width and height is equal to the demand d_j of that p_j with the same width and height. Thus, $|P'| = \sum_{j=1}^{n} d_j$

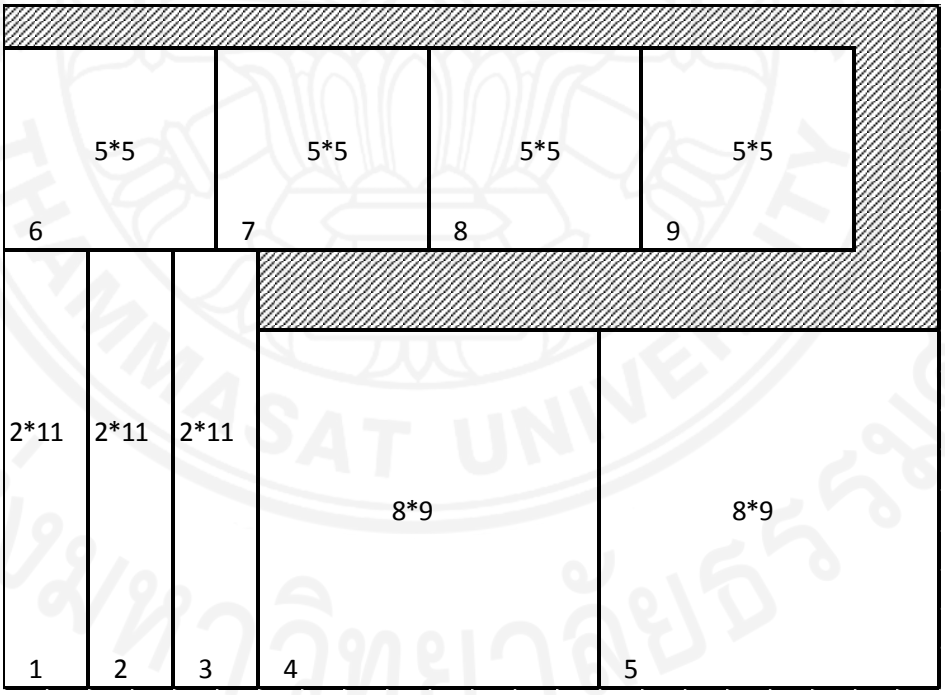
Step 3: Each $p'_j \in P'$, selected p'_j to put into the space at the bottom left of the sheet $s_i \in S$. Then, next panel $p'_j \in P'$ is selected to put into the space next to the previous panel on the same strip $s_i \in S$ in horizontal line until no more panels types can be inserted in that strip, then move to next horizontal strip. If there are not any strips available, it moves to the new sheet.

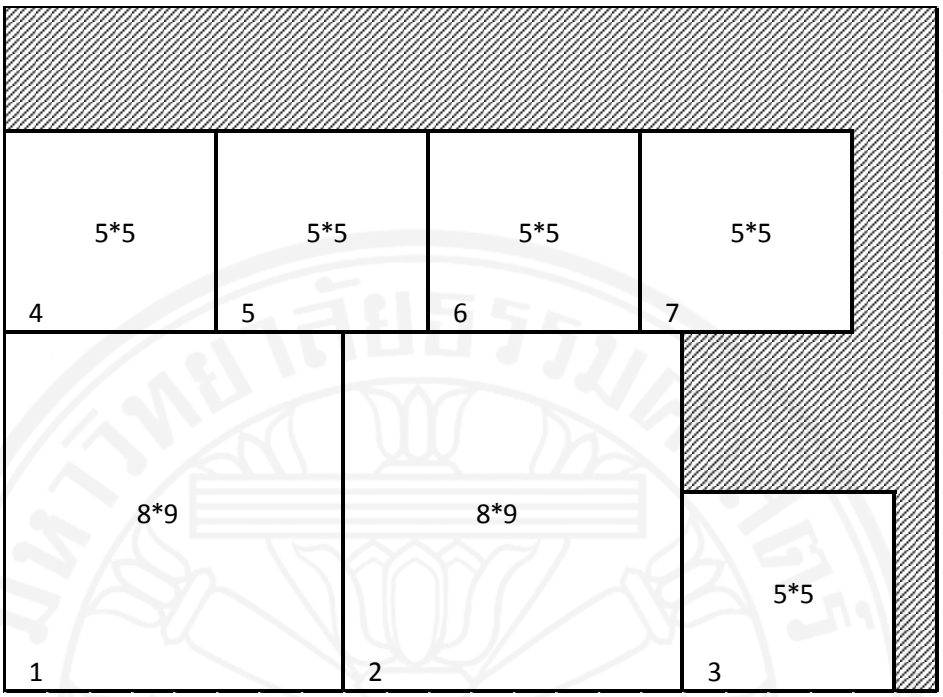
Step 4: Remove $p'_j \in P'$ that has been put.

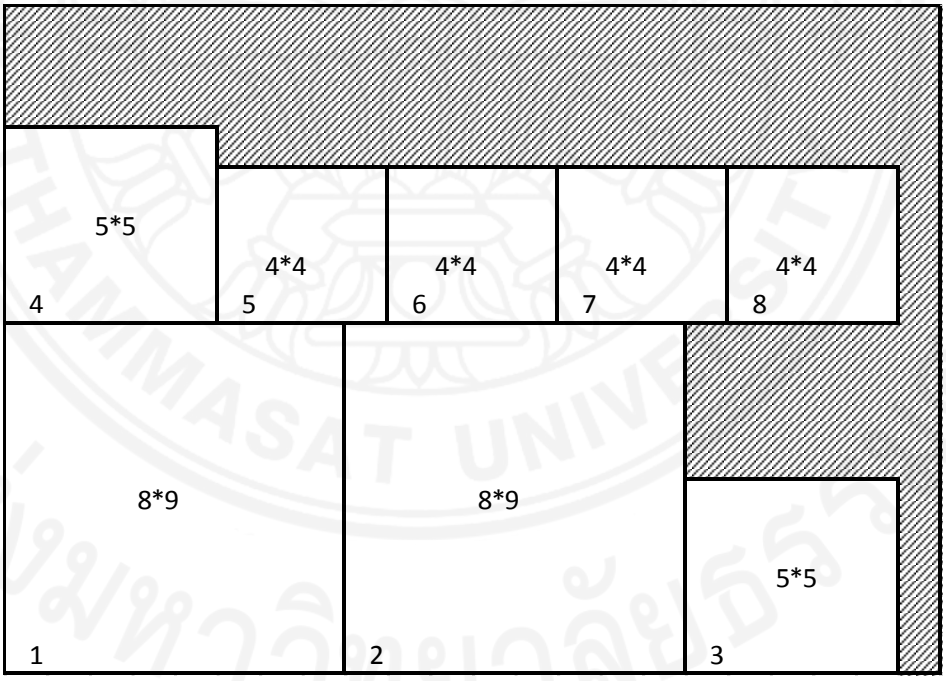
Step 5: If $P' = \{0\}$, end the cutting process; otherwise, go to step 3

From the same instance mentioned in section 3.2, total number of sheet ought to cut for each pattern as illustrated in Figure 3.6 is 1, 1, 2, 1, 1, and 1 respectively. The total waste after satisfy all the demands is 674 in^2.


5*5
12
5*5
13
5*5
14
5*5
15
2*11
1
2*11
2
2*11
3
2*11
4
2*11
5
2*11
6
2*11
7
2*11
8
2*11
9
2*11
10
2*11
11


5*5
6
5*5
7
5*5
8
5*5
9
2*11
1
2*11
2
2*11
3
8*9
4
8*9
5


5*5
5*5
5*5
5*5
4
5
6
7
8*9
8*9
5*5
1
2
3


5*5
4*4
4*4
4*4
4*4
4
5
6
7
8
8*9
8*9
5*5
1
2
3

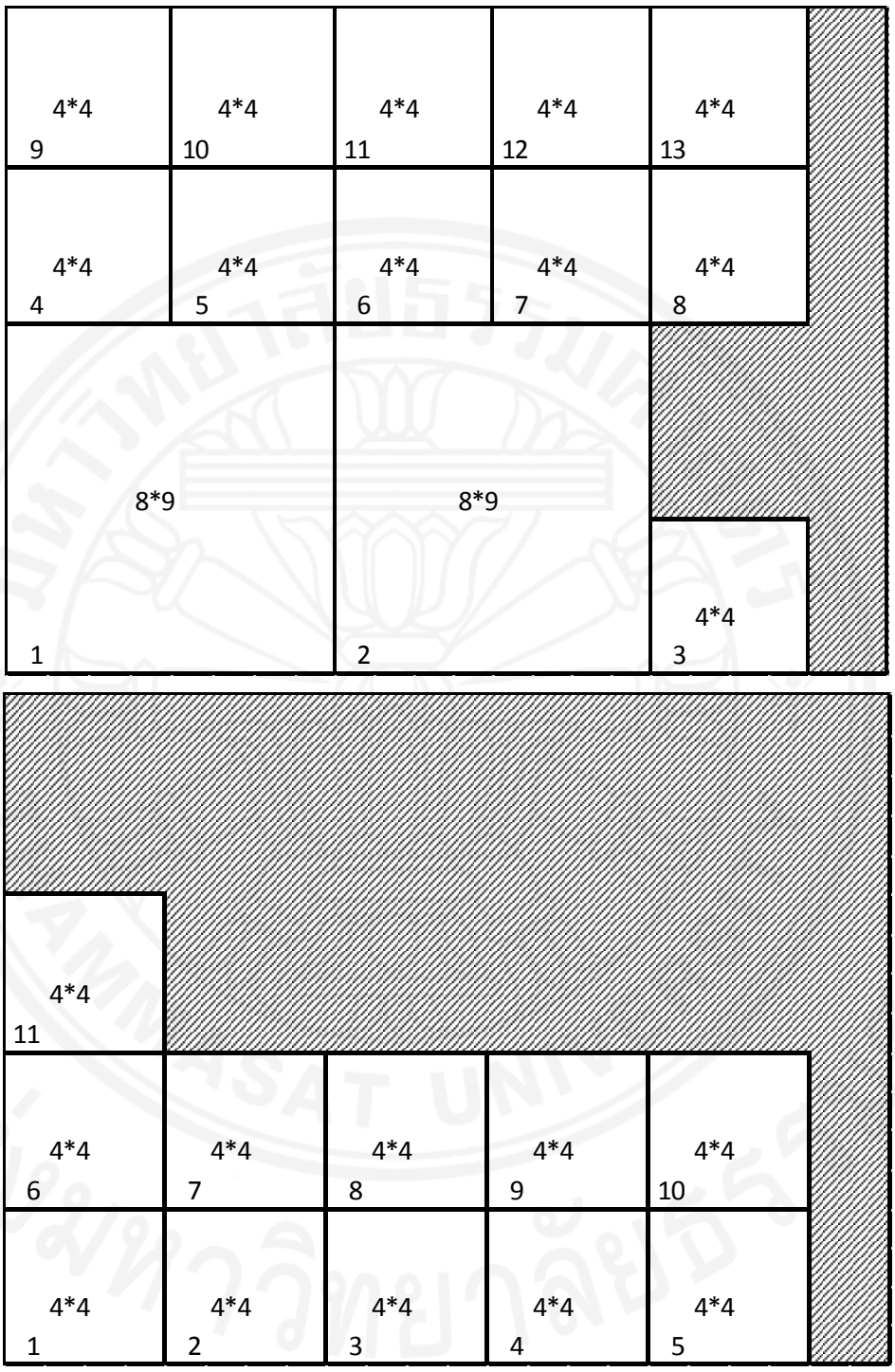


Figure 3.6 Solution of SW_PH_HC

3.8. Sheet Width Panel Width Vertical Cut (SW_PW_VC)

In this section, the panels are arranged to cut vertically. Multiple types of panels are allowed to put in each size of sheet. The process of this method is implemented as in the following.

Step 1: Given a set of sheets $S = \{s_1, s_2, ..., s_i\}$ and a set of panels $P = \{p_1, p_2, ..., p_j\}$. Then sort the panels and the sheets based on the width in descending order.

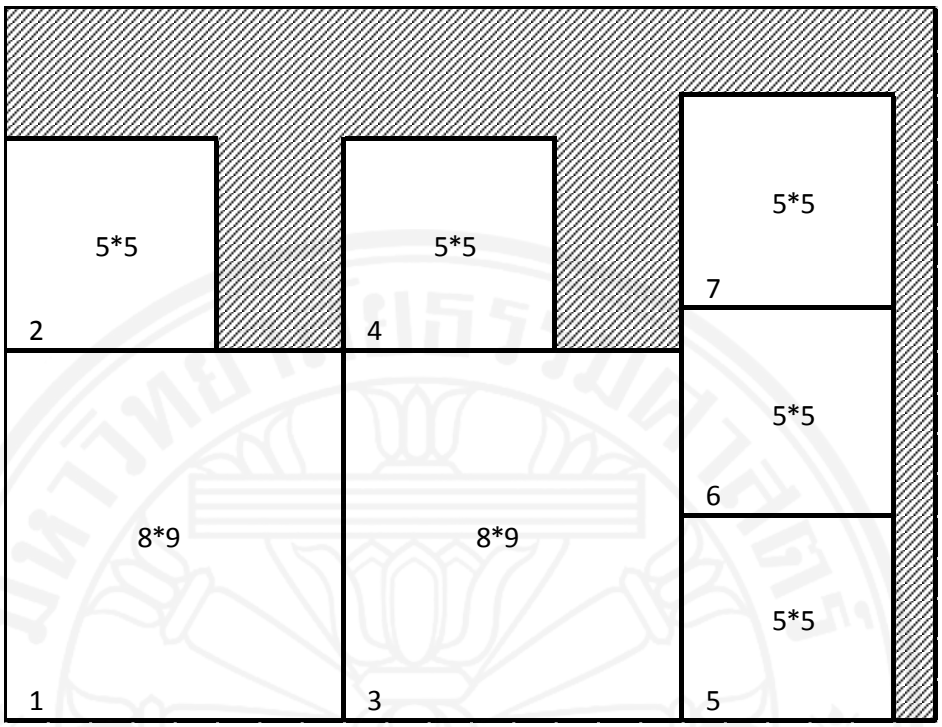
Step 2: P' is the set of panel where $p'_j \in P'$, such that the number of p'_j with the same width and height is equal to the demand d_j of that p_j with the same width and height. Thus, $|P'| = \sum_{j=1}^{n} d_j$

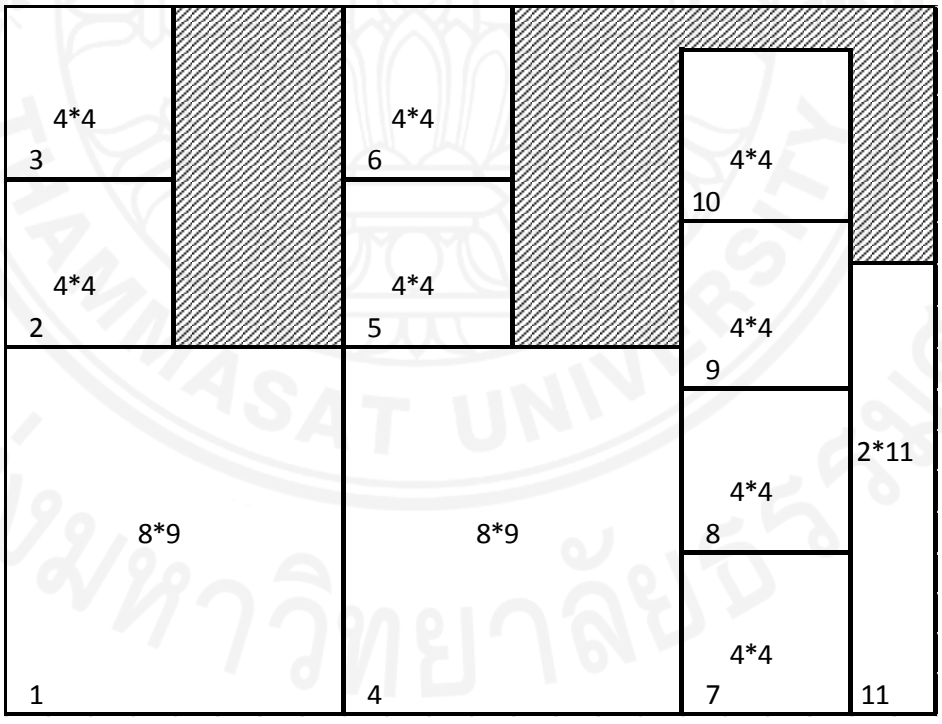
Step 3: Each $p'_j \in P'$, selected p'_j to put into the space at the bottom left of the sheet $s_i \in S$. Then, next panels is selected to put into the space next to the previous panel on the same strip $s_i \in S$ in vertical line until no more panels types can be inserted in that vertical strip, then move to next strip to cut vertically. If there are not any strips available, move to the new sheet.

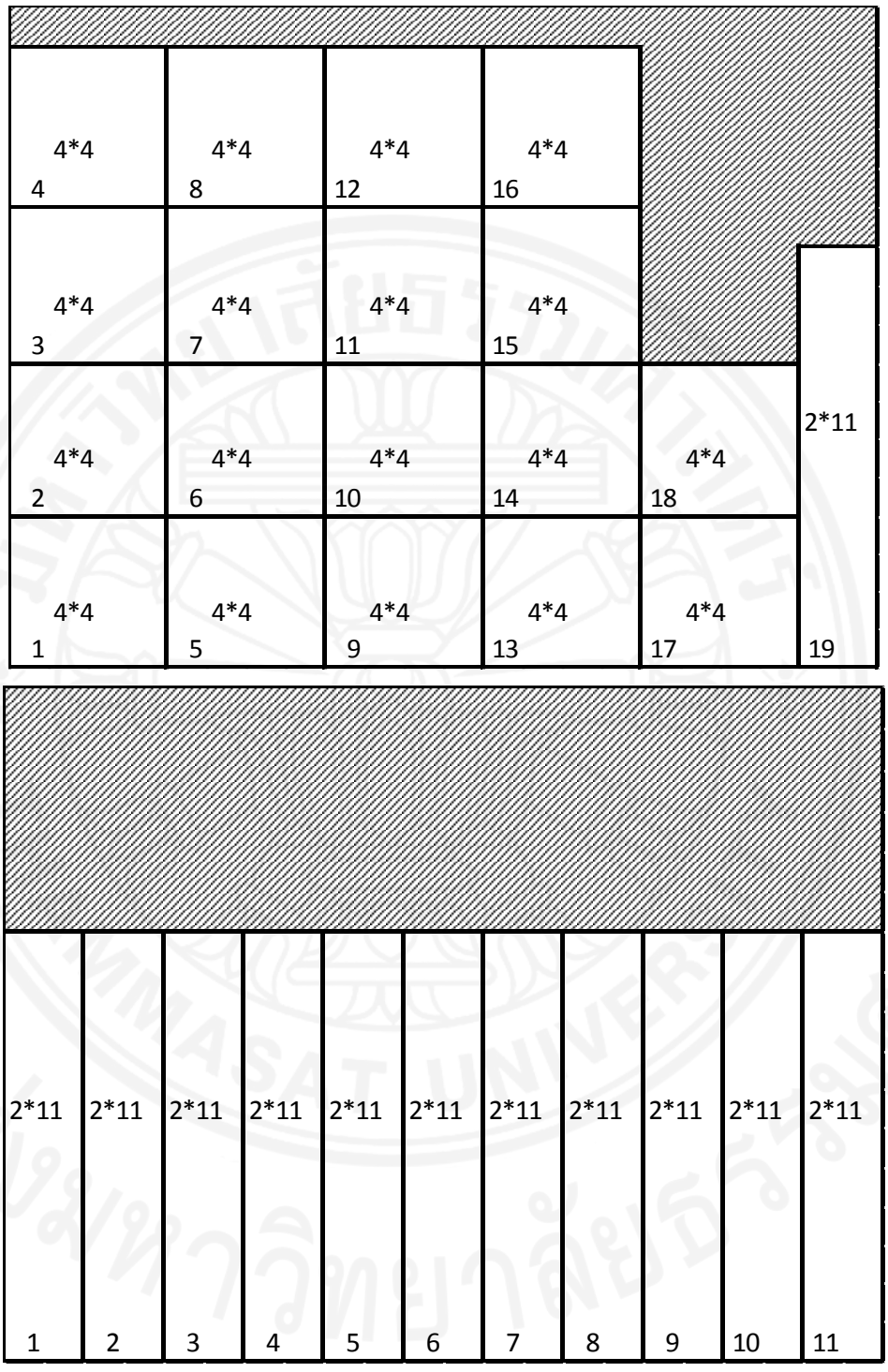
Step 4: Remove $p'_j \in P'$ that has been put.

Step 5: If $P' = \{0\}$, end the cutting process; otherwise, go to step 3

From the same instance mentioned in section 3.2, the number of sheet, cut for each patterns as illustrated in Figure 3.7, are 4, 1, 1, 1, and 1 respectively. The total waste after fulfill all the demands is 1048 in^2


5*5
2
5*5
4
5*5
7
5*5
6
8*9
1
8*9
3
5*5
5


4*4
3
4*4
6
4*4
10
4*4
2
4*4
5
4*4
9
2*11
4*4
8
8*9
1
8*9
4
4*4
7
11


4*4
4
4*4
8
4*4
12
4*4
16
4*4
3
4*4
7
4*4
11
4*4
15
4*4
2
4*4
6
4*4
10
4*4
14
4*4
18
2*11
4*4
1
4*4
5
4*4
9
4*4
13
4*4
17
19
2*11
1
2*11
2
2*11
3
2*11
4
2*11
5
2*11
6
2*11
7
2*11
8
2*11
9
2*11
10
2*11
11

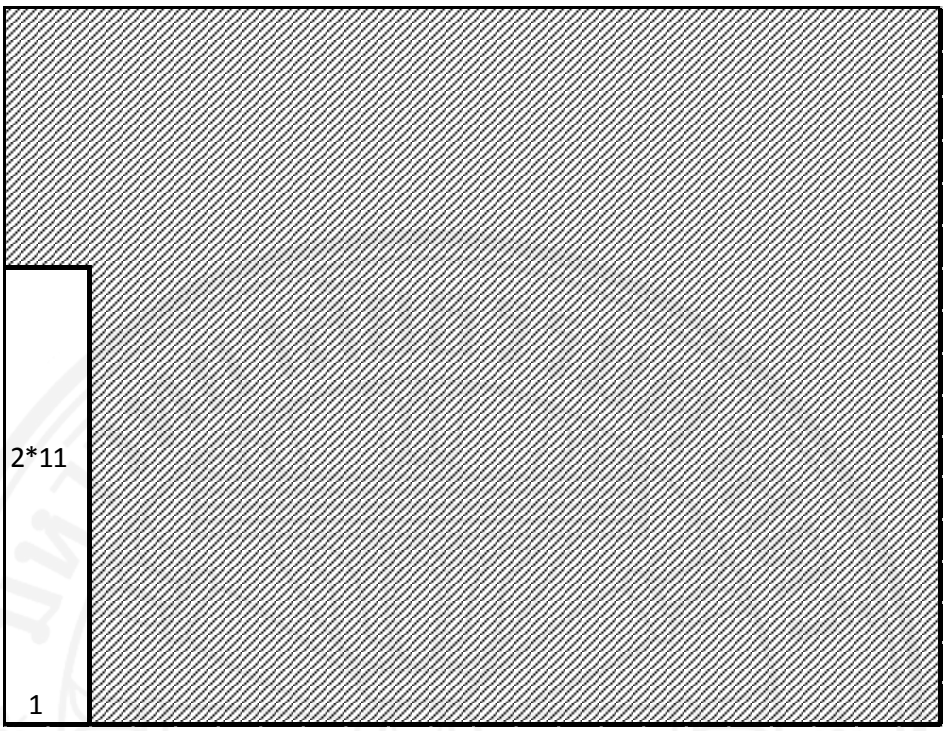


Figure 3.7 Solution of (SW_PW_VC)

3.9. Minimum Sheet Width Ordering Panel Height Horizontal Cut (MinSW_OPH_HC)

In this section, the panels are arranged to cut horizontally. Multiple types of panels are allowed to put into each size of sheet. The process of this method is mentioned as in the following.

Step 1: Given a set of sheets $S = \{s_1, s_2, ..., s_i\}$ and a set of panels $P = \{p_1, p_2, ..., p_j\}$. Then sort the panel set in descending order based on height. P' is also the set of panel where $p'_j \in P'$, such that the number of p'_j with the same width and height is equal to the demand d_j with the same width and height. Thus, $|P'| = \sum_{j=1}^{n} d_j$.

Step 2: Select $p'_j \in P'$, put p'_j in each sheet $s_i \in S$ at the bottom left corner in horizontal line for the first strip. Then calculate the leftover width after putting the first panel. Take the leftover width divide with each type of panel width. The panels that can give the minimum division remainder is selected to put close to the first panel. This process is repeated until no more panels can be filled up in the first strip. Once the

horizontal strip is filled up, move to the next strip vertically and repeat the same process until there is no more space to place any panels $p'_j \in P'$.

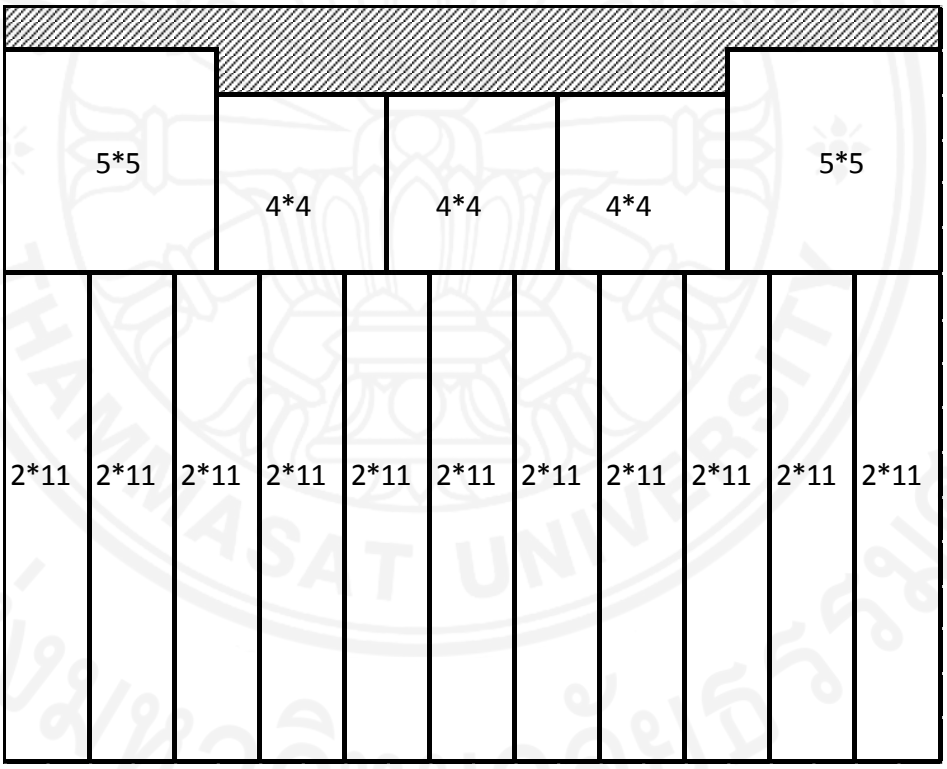
Step 3: Select $s_i \in S$ are used in step 2 that can give a minimum waste to cut.

Step 4: Number of the sheet ought to cut is equal to the ceiling of the smallest ratio between the demands and the number of each panel type used in the pattern

Step 5: Remove $p'_j \in P'$ that has been put.

Step 6: If $P' = \{0\}$, end the cutting process; otherwise, go to step 2

From the same instance mentioned in section 3.2, total number of sheet, needs to cut for each pattern as illustrated in Figure 3.8, are 2, 4, 1, and 1 respectively. The total waste after fulfill all the demands is 1278 in^2




8*9
5*5
8*9
5*5
8*9
5*5
5*5
4*4
5*5
5*5
4*4
5*5
5*5
4*4
5*5
5*5
4*4
5*5
5*5
4*4
5*5
5*5
4*4
5*5

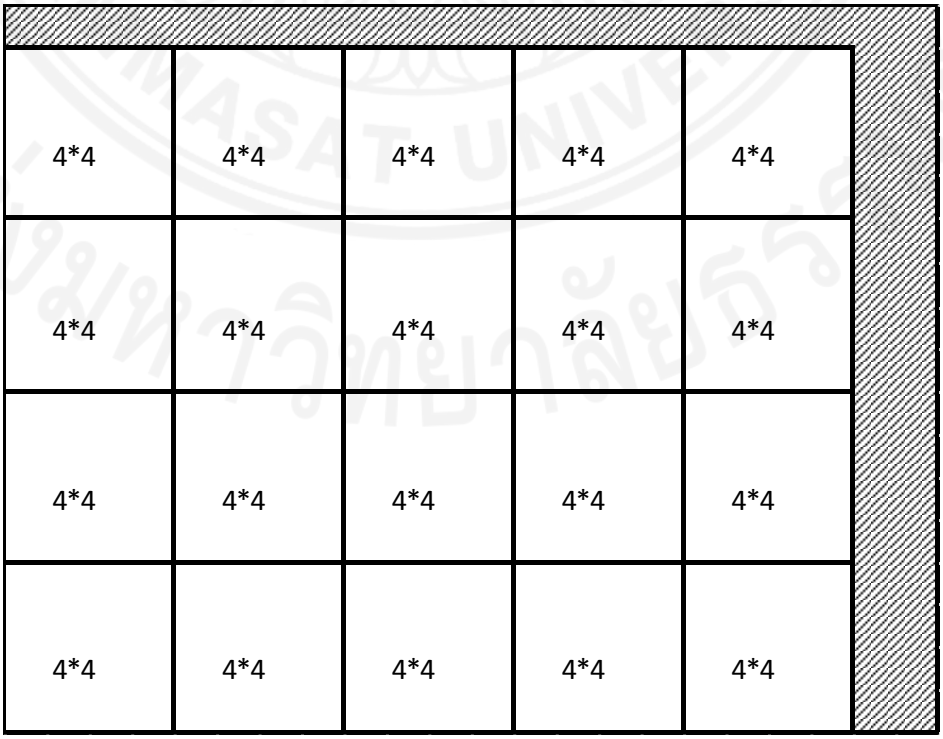

4*4
4*4
4*4
4*4
4*4
4*4
4*4
4*4
4*4
4*4
4*4
4*4
4*4
4*4
4*4
4*4
4*4
4*4
4*4
4*4

Figure 3.8 Solution of (MinSW_OPH_HC)

3.10. Minimum Sheet Height Ordering Panel Width Vertical Cut (MinSH_OPW_VC)

In this section, the panels are arranged and cut vertically. Multiple types of panels are allowed to put in each size of sheet. The process of this method is mentioned as in the following.

Step 1: Given a set of sheets $S = \{s_1, s_2, ..., s_i\}$ and a set of panels $P = \{p_1, p_2, ..., p_j\}$. Then sort the panel set in descending order based on width. P' is also the set of panel where $p'_j \in P'$, such that the number of p'_j with the same width and height is equal to the demand d_j with the same width and height. Thus, $|P'| = \sum_{j=1}^{n} d_j$.

Step 2: Select $p'_j \in P'$, put p'_j in each sheet $s_i \in S$ at the bottom left corner in vertical line for the first strip. Then calculate the leftover height after putting the first panel p'_j. Take the leftover height divide with each type of panel height. The panels, that can give the minimum division remainder, is selected to put above the first panel. This process is repeated until no more panels can be filled up in the first strip. Once the vertical strip is filled up, move to the next strip horizontally and repeat the same process until there is no more space to place other panels $p'_j \in P'$.

Step 3: Select $s_i \in S$ are used in step 3 that can give a minimum waste to cut.

Step 4: Number of the sheet ought to cut is equal to the ceiling of the smallest ratio between the demands and the number of each panel type put in the pattern

Step 5: Remove $p'_j \in P'$ that has been put.

Step 6: If $P' = \{0\}$, end the cutting process; otherwise, go to step 3

From the same instance mention in section 3.2, the number of sheet, needs to cut for each pattern as illustrated in Figure 3.9, are 4, 2, and 1 respectively. The total waste after fulfill all the demands is 812 in^2.

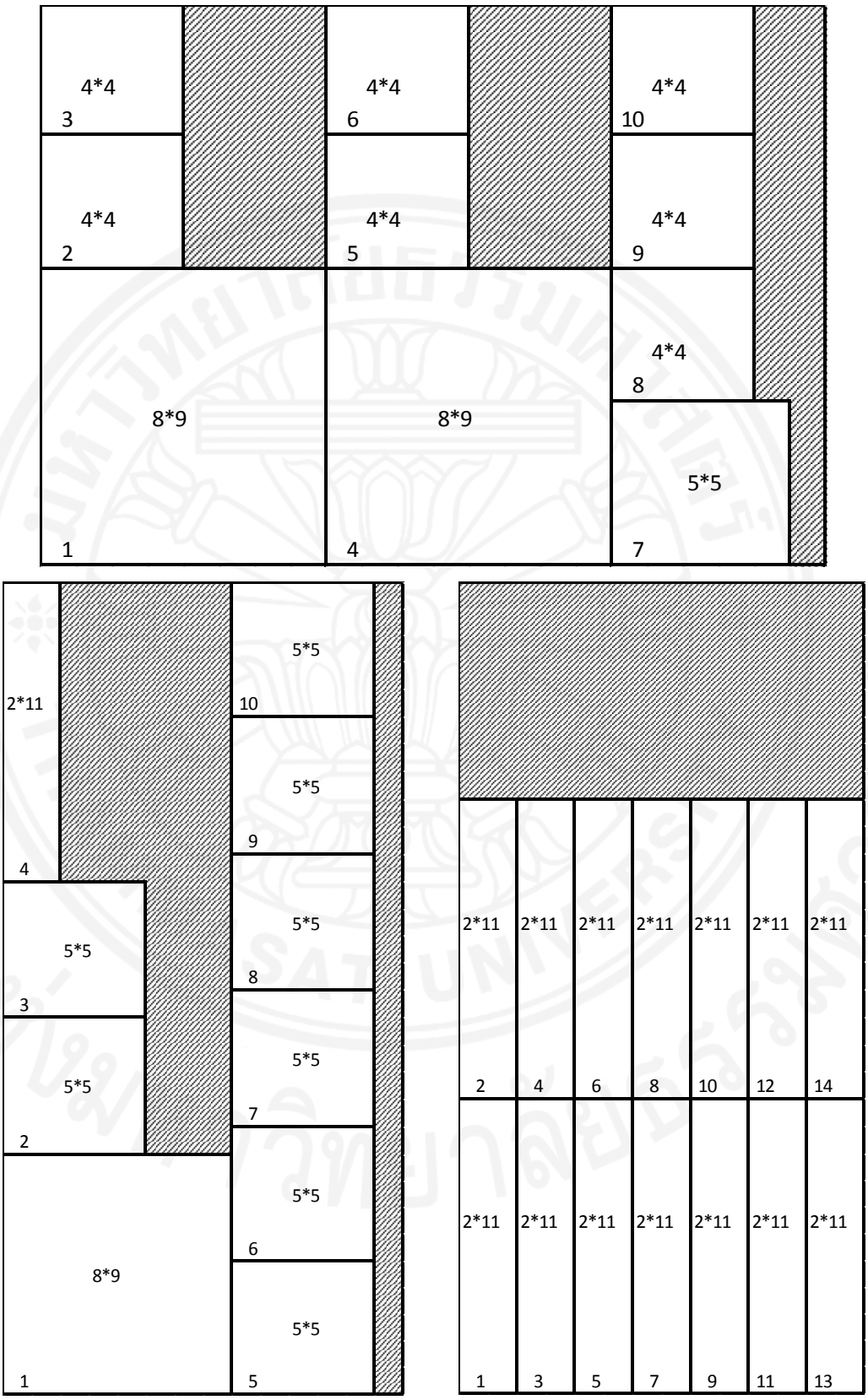


Figure 3.9 Solution of (MinSH_OPW_VC)

Chapter 4
Testing Instances

Table 4.1 illustrated a list of twenty different size of instances. These instances are tested with ten techniques mentioned in section 3. Total instances are grouped into three categories in accordance with the total number of sheet types and panel types. If total different type of sheets and panels is less than 6, it is regarded as small size instances. If it is from 7 to 20, it is regarded as medium size instances. It is a large size instance when it is bigger than 20. CG is used as a benchmark to compare with other 9 heuristic techniques. In addition, 2DSHC is used to compare with other 8 heuristic methods as well since it is a simple cutting patterns that applied by electronic board cutting industrial. Both total waste and the computational time of each instance are compared between each technique.

Table 4.1 The input for small, medium, and large size instances

Size	No	#Sheet type	#Panel type	#Demand
Small Size	1	2	2	219
	2	2	2	301
	3	2	2	575
	4	2	3	537
	5	3	3	552
	6	4	2	300
	7	5	1	209
Medium Size	8	5	6	1,492
	9	5	8	2,240
	10	6	2	1,329
	11	6	3	219
	12	8	7	596
	13	9	3	2,192
	14	9	9	3,068
	15	11	4	4,299
	16	11	5	748
Large Size	17	15	12	4,429
	18	16	6	1,121
	19	17	27	10,611
	20	26	15	3,790

Chapter 5
Experimental Result and Discussion

In order to evaluate the performance of these techniques, we run it on Intel® Core™ i5-4200U CPU @ 1.60GHz 2.30GHz, installed memory (RAM) 4.00GB machine running under the Windows environment. CG technique is coded in IBM ILOG CPLEX Optimization Studio (64bit) 12.6, and the rest of the techniques are implemented on JAVA 7. All type of technique is used to compare with the CG technique based on the gap given by:

$$Gap(\%) = \frac{waste\,of\ each\ technique - waste\ of\ CG}{waste\ of\ CG} \times 100$$

5.1. Comparison to column generation

For 2DSHC, 3 instances provide same amount of total waste, 17 instances provide higher total amount of waste for 2DSHC as illustrated in Table 5.1. This method is not really good at all to find pattern to cut since it allows only one panel type to put in each size of sheet. However, the patterns created is simple and easy to cut.

2DHC cannot provide any better solutions at all. It is even worse for medium and large size of instances as illustrated in Table 5.1. This method cannot find a good pattern. When a sheet set and a panel set are sorted in order from the highest to the shortest based on height, only the first sheet is selected to cut since each sheet has unlimited capacity in stock. For instance 15, the gap of waste goes up to 24,344.5%. This gap might be reduced, if other sheet type is selected.

2DVC also cannot provide any better solutions at all. It is even worse for medium and large size of instances as illustrated in Table 5.1. This method cannot find a good pattern. When a sheet set and a panel set are sorted in order from the highest to the shortest based on height and width respectively, only the first sheet is selected to cut since each sheet has unlimited capacity in stock. For instance 15, the gap of waste goes up to 23,302.8%. This gap might be reduced, if other sheet type is selected.

2DHI come from a combination between 2DSHC and 2DHC. 5 instances provide same amount of waste, 11 instances provide a higher waste, and 4 instances provide lower wastes as illustrated in Table 5.1. This technique is better than 2DHC because it can explore additional sheets with a minimum waste to cut, and more than one panel type is allowed to put into each types of sheet if the space is large enough.

2DVI come from a combination between 2DSHC and 2DVC. 5 instances provide same amount of waste, 11 instances provide higher waste, and 4 instances provide lower wastes as illustrated in Table 5.1. This technique is better than 2DHC because it can explore additional sheets with a minimum waste to cut, and more than one panel type is allowed to put into each types of sheet if the space is large enough.

SW_PH_HC cannot provide good solution. 2 instances provide same amount of waste, and other 18 instances provide bad pattern with high amount of waste as illustrated in Table 5.2. When a sheet set is sorted based on width, and a panel set is sorted based on height from the highest one to the shortest one, only the first sheet is selected to cut since each sheet has unlimited capacity in stock.

SW_PW_VC cannot provide good solution. 2 instances provide same amount of waste, and other 18 instances provide bad pattern with high amount of waste as illustrated in Table 5.2. When a sheet set and a panel set are sorted based on width from the highest one to the shortest one, only the first sheet is selected to cut since each sheet has unlimited capacity in stock.

MinSW_OPH_HC provides same amount of waste for 3 instances. 15 instances provide higher waste, and 2 instances get lower wastes or can find a better pattern as illustrated in Table 5.2. Actually, after a panel set is sorted in descending order in accordance with height, only the panel type provided a minimum sheet width is selected to put from iteration to iteration to cut horizontally. This reason can make the given pattern good, but data contain in each tested instances maybe not appropriate to use this technique.

MinSH_OPW_VC provides same amount of waste for 5 instances. 11 instances provide higher waste, and 4 instances get lower wastes or can find a better pattern as illustrated in Table 5.2. After a panel set is sorted in descending order in accordance with width, only the panel type provided a minimum sheet height is selected to put from iteration to iteration to cut horizontally. This reason can make the given

pattern good. Even some instances provide higher amount of waste, the gap is not much different.

From this comparison in Table 5.1 and Table 5.2, four out of nine techniques can provide better solution in some instances. They are 2DHI, 2DVI, MinSW_OPH_HC, and MinSH_OPW_VC. These four technique are compared with each other in Table 5.3.

5.2. Comparison to 2D Simple Heuristic Cutting

2DHC can provide a better pattern for 1 instance, same pattern for 1 instance, and other 18 instances cannot provide a better solution as illustrated in Table 5.1.

2DVC can provide a better pattern for 1 instance, and other 19 instances cannot provide a better solution as illustrated in Table 5.1.

For 2DHI, 7 instances can provide the same amount of waste, 12 instances can provide a better pattern, and only 1 instance that provide a bad pattern as illustrated in Table 5.1.

For 2DVI, 9 instances can provide the same amount of waste, 11 instances can provide a better pattern as illustrated in Table 5.1.

For SW_PH_HC, 6 instances provide a better pattern, 14 instances provide a worse patterns as illustrated in Table 5.2.

SW_PW_VC, 4 instances provide a better pattern, 16 instances provide a worse patterns as illustrated in Table 5.2.

For MinSW_OPH_HC, 3 instances provide the same amount of waste, 7 instances provide higher amount of waste, and 10 instances provide lower amount of wastes as illustrated in Table 5.2.

For MinSH_OPW_VC, 6 instances provide the same amount of waste, and the other 14 instances provide lower amount of waste as illustrated in .

From a comparison to 2D simple heuristic cutting we found that all proposed technique is not bad. There are at least a few instances that can provide better solution. Most of instances canTable 5.2 find a good pattern in 2DHI, 2DVI, MinSW_OPH_HC, and MinSH_OPW_VC technique.

Table 5.1 The output of CG, 2DSHC, 2DHC, 2DVC, 2DHI, and 2DVI including waste, time, and gap

Size	No	Number of			CG		2DSHC			2DHC			2DVC			2DHI			2DVI		
		Sheet	Panel	Demand	Waste	Time	Waste	Time	Gap	Waste	Time	Gap	Waste	Time	Gap	Waste	Time	Gap	Waste	Time	Gap
Unit	0	(type)	(type)	(panel)	(Sq In)	(Sec)	(Sq In)	(Sec)	(%)	(Sq In)	(Sec)	(%)	(Sq In)	(Sec)	(%)	(Sq In)	(Sec)	(%)	(Sq In)	(Sec)	(%)
Small	1	2	2	219	58,228.90	1.2	58,228.90	1.3	0	59,362.90	1.4	1.9	59,362.90	1.4	1.9	58,228.90	1.4	0	58,228.90	1.3	0
	2	2	2	301	115,364.0	1.8	115,364.40	1.3	0	115,364.40	1.4	0	144,356.40	1.6	25.1	115,364.40	1.4	0	115,364.40	1.3	0
	3	2	2	575	98,370.80	3.2	185,445.80	1.4	88.5	252,918.80	1.8	157.1	252,918.80	1.6	157.1	98,370.80	1.3	0	98,370.80	1.3	0
	4	2	3	537	35,124.90	4.5	94,455.90	1.3	169	60,183.90	1.6	71.3	60,183.90	1.6	71.3	52,491.90	1.3	49.4	36,624.90	1.3	4.3
	5	3	3	552	29,108.20	9.2	29,108.20	1.4	0	128,180.20	1.5	340.4	128,180.20	1.6	340.4	29,108.20	1.4	0	29,108.20	1.4	0
	6	4	2	300	31,800.00	3.4	34,224.00	1.3	7.6	31,800.00	1.4	0	31,800.00	1.5	0	34,224.00	1.2	7.6	34,224.00	1.3	7.6
	7	5	1	209	70,101.60	1	70,376.60	1.3	0.4	75,626.60	1.5	7.9	75,626.60	1.4	7.9	70,376.60	1.2	0.4	70,376.60	1.3	0.4
Medium	8	5	6	1,492	274,683.0	66.8	558,711.60	1.4	103	1,534,059.6	2.1	458.5	2,058,123.6	1.9	649.3	274,904.40	1.5	0.1	558,711.60	1.5	103
	9	5	8	2,240	93,677.40	68	95,692.20	1.7	2.2	713,145.00	1.8	661.3	713,145.00	2.2	661.3	95,692.20	1.5	2.2	95,692.20	1.5	2.2
	10	6	2	1,329	194,561.0	28.4	351,808.50	1.5	80.8	361,528.50	1.6	85.8	361,528.50	1.8	85.8	192,808.50	1.3	-0.9	192,808.50	1.4	-0.9
	11	6	3	219	7,576.90	5.5	9,976.90	1.3	31.7	41,200.90	1.4	443.8	41,200.90	1.4	443.8	7,480.90	1.4	-1.3	9,976.90	1.3	31.7
	12	8	7	596	42,051.00	35	43,785.00	1.4	4.1	54,561.00	1.6	29.7	54,561.00	1.6	29.7	43,785.00	1.5	4.1	43,785.00	1.3	4.1
	13	9	3	2,192	115,340.0	88.8	310,399.60	1.5	169	322,483.60	2	179.6	322,483.60	2	179.6	256,366.60	1.5	122	98,779.60	1.5	-14.4
	14	9	9	3,068	963,234.0	205.5	1,270,337.8	1.6	31.9	1,473,185.8	2.1	52.9	1,473,185.8	2.3	52.9	1,262,081.8	1.6	31	1,264,961.8	1.6	31.3
	15	11	4	4,299	8,694.90	545.4	18,300.50	1.6	111	2,125,422.5	2.4	24,344.5	2,034,846.5	2.7	23,302.8	10,954.00	1.6	26	15,556.50	1.6	78.9
	16	11	5	748	28,908.00	67.9	51,962.50	1.5	79.8	113,155.00	1.6	291.4	113,155.00	1.7	291.4	28,908.00	1.4	0	28,908.00	1.4	0
Large	17	15	12	4,429	68,363.40	472.3	167,909.30	1.7	146	2,347,191.3	2.2	3,333.40	2,350,047.3	2.9	3,337.60	70,390.60	1.6	3	67,550.30	1.8	-1.2
	18	16	6	1,121	31,768.90	85.8	36,019.90	1.6	13.4	161,050.40	1.7	406.9	161,050.40	1.8	406.9	39,751.60	1.5	25.1	33,475.90	1.4	5.4
	19	17	27	10,611	451,778.8	1,467.00	567,036.00	2.1	25.5	943,605.60	3.3	108.9	971,685.60	3.8	115.1	371,064.50	2.2	-17.9	510,624.00	2.9	13
	20	26	15	3,790	149,754.0	869.4	189,450.20	1.9	26.5	365,469.20	2.2	144	1,218,523.2	2.6	713.7	141,894.20	1.7	-5.2	134,174.40	1.7	-10.4

Table 5.2 The output of CG, 2DSHC, 2DHC_SW_PH, 2DVC_SW_PW, MinSW_OPH_HC, and MinSH_OPW_VC including waste, time, and gap

Size	No	Number of			CG		2DSHC			SW_PH_HC			SW_PW_VC			MinSW_OPH_HC			MinSH_OPW_VC		
		Sheet	Panel	Demand	Waste	Time	Waste	Time	Gap	Waste	Time	Gap	Waste	Time	Gap	Waste	Time	Gap	Waste	Time	Gap
Unit	0	(type)	(type)	(panel)	(Sq In)	(Sec)	(Sq In)	(Sec)	(%)	(Sq In)	(Sec)	(%)	(Sq In)	(Sec)	(%)	(Sq In)	(Sec)	(%)	(Sq In)	(Sec)	(%)
Small	1	2	2	219	58,228.90	1.2	58,228.90	1.3	0	59362.905	1.27	1.9	59362.905	1.18	1.9	114949.905	2.177	97.4	58228.905	1.069	0
	2	2	2	301	115,364.00	1.8	115,364.40	1.3	0	144356.4	1.21	25.1	144356.4	1.2	25.1	115364.4	1.076	0	115364.4	1.04	0
	3	2	2	575	98,370.80	3.2	185,445.80	1.4	88.5	98370.75	1.43	0	98370.75	1.3	0	98370.75	1.127	0	98370.75	1.059	0
	4	2	3	537	35,124.90	4.5	94,455.90	1.3	169	60183.888	1.25	71.3	60183.888	1.26	71.3	52491.8875	1.137	49.4	36624.8875	1.12	4.3
	5	3	3	552	29,108.20	9.2	29,108.20	1.4	0	128180.17	1.22	340	128180.17	1.29	340	29108.17	1.164	0	29108.17	1.062	0
	6	4	2	300	31,800.00	3.4	34,224.00	1.3	7.6	31800	1.17	0	31800	1.18	0	123112	1.184	287	34224	1.048	7.6
	7	5	1	209	70,101.60	1	70,376.60	1.3	0.4	75136.6	1.14	7.2	75136.6	1.19	7.2	70376.6	1.096	0.4	70376.6	1.116	0.4
Medium	8	5	6	1,492	274,683.00	66.8	558,711.60	1.4	103	338824.09	1.64	23.4	660509.09	1.44	141	471173.09	1.345	71.5	385986.59	1.123	40.5
	9	5	8	2,240	93,677.40	68	95,692.20	1.7	2.2	713145	1.53	661	713145	1.46	661	135685.8	1.556	44.8	95692.2	1.132	2.2
	10	6	2	1,329	194,561.00	28.4	351,808.50	1.5	80.8	430068	1.53	121	430068	1.33	121	192808.5	1.137	-0.9	192808.5	1.096	-0.9
	11	6	3	219	7,576.90	5.5	9,976.90	1.3	31.7	41200.9	1.16	444	41200.9	1.19	444	7480.9	1.101	-1.3	7480.9	1.079	-1.3
	12	8	7	596	42,051.00	35	43,785.00	1.4	4.1	54561	1.34	29.7	54561	1.55	29.7	42705	1.173	1.6	42705	1.099	1.6
	13	9	3	2,192	115,340.00	88.8	310,399.60	1.5	169	322483.58	1.66	180	322483.58	1.47	180	119491.583	1.161	3.6	98779.5825	1.126	-14
	14	9	9	3,068	963,234.00	205.5	1,270,337.80	1.6	31.9	1473185.8	1.59	52.9	1473185.8	1.64	52.9	1262081.75	1.213	31	1264961.75	1.154	31.3
	15	11	4	4,299	8,694.90	545.4	18,300.50	1.6	111	13134.5	1.55	51.1	93102.5	1.47	971	24684.5	1.376	184	10954	1.157	26
	16	11	5	748	28,908.00	67.9	51,962.50	1.5	79.8	110069.5	1.3	281	110069.5	1.29	281	78821.5	1.176	173	28908	1.201	0
Large	17	15	12	4,429	68,363.40	472.3	167,909.30	1.7	146	113533.29	1.65	66.1	113533.29	1.63	66.1	69581.7875	1.396	1.8	67666.7875	1.277	-1
	18	16	6	1,121	31,768.90	85.8	36,019.90	1.6	13.4	161050.43	1.38	407	161050.43	1.4	407	91780.925	1.191	189	33475.925	1.19	5.4
	19	17	27	10,611	451,778.80	1,467.00	567,036.00	2.1	25.5	943605.6	2.13	109	971685.6	2.33	115	914136.603	1.956	102	511298.403	1.336	13.2
	20	26	15	3,790	149,754.00	869.4	189,450.20	1.9	26.5	365469.16	1.67	144	365469.16	1.6	144	178039.66	1.376	18.9	153178.16	1.263	2.3

5.3. Comparison of computational time

In CG technique, when the size of the problems are larger and larger, the computational time increases exponentially. The pattern can be provided in a sort computational time even for large size instances for 2DSHC and all proposed heuristic techniques.

5.4. Comparison of 2DHI, 2DVI, MinSW_OPH_HC, MinSH_OPW_VC

Four techniques namely: 2DHI, 2DVI, MinSW_OPH_HC, and MinSH_OPW_VC are compared with each other in term of waste as illustrated in Table 5.3. Each of these four techniques provides a good pattern for a few instances. It is better than column generation, and many instances is better than 2D simple heuristic cutting. Among these four techniques, MinSW_OPH_HC can provide only 2 instances which is better than the column generation while the other three techniques can provide up to 4 instances for each technique.

It would be great to combine 2DHI with 2DVI or MinSW_OPW_VC because the total number of instances provide a better pattern will go up to 6 instances. Even some instances cannot provide better pattern, but those patterns do not provide a large area of waste at all. To the best of our knowledge, 2DHI, 2DVI, MinSW_OPH_HC, and MinSH_OPW_VC are not overlapped each other. It depends on the size of each panel type and sheet type in each instance.

Table 5.3 Waste comparison of 2DHI, 2DVI, MinSW_OPH_HC, and MinSW_OPW_VC

Size	No	Number of			CG	2DHI		2DVI		MinSW_OPH_HC		MinSH_OPW_VC	
		Sheet	Panel	Demand	Waste	Waste	Gap	Waste	Gap	Waste	Gap	Waste	Gap
Unit	0	(type)	(type)	(panel)	(Sq In)	(Sq In)	(%)	(Sq In)	(%)	(Sq In)	(%)	(Sq In)	(%)
Small	1	2	2	219	58,228.90	58,228.91	0	58,228.91	0	114949.905	97.41	58228.905	0
	2	2	2	301	115,364.00	115,364.40	0	115,364.40	0	115364.4	0	115364.4	0
	3	2	2	575	98,370.80	98,370.75	0	98,370.75	0	98370.75	0	98370.75	0
	4	2	3	537	35,124.90	52,491.89	49.44	36,624.89	4.27	52491.8875	49.44	36624.8875	4.27
	5	3	3	552	29,108.20	29,108.17	0	29,108.17	0	29108.17	0	29108.17	0
	6	4	2	300	31,800.00	34,224.00	7.62	34,224.00	7.62	123112	287.1	34224	7.62
	7	5	1	209	70,101.60	70,376.60	0.39	70,376.60	0.39	70376.6	0.39	70376.6	0.39
Medium	8	5	6	1,492	274,683.00	274,904.38	0.08	558,711.59	103.4	471173.09	71.53	385986.59	40.52
	9	5	8	2,240	93,677.40	95,692.20	2.15	95,692.20	2.15	135685.8	44.84	95692.2	2.15
	10	6	2	1,329	194,561.00	192,808.50	-0.9	192,808.50	-0.9	192808.5	-0.9	192808.5	-0.9
	11	6	3	219	7,576.90	7,480.90	-1.27	9,976.90	31.68	7480.9	-1.27	7480.9	-1.27
	12	8	7	596	42,051.00	43,785.00	4.12	43,785.00	4.12	42705	1.56	42705	1.56
	13	9	3	2,192	115,340.00	256,366.58	122.3	98,779.58	-14.4	119491.583	3.6	98779.5825	-14.4
	14	9	9	3,068	963,234.00	1,262,081.75	31.03	1,264,961.75	31.32	1262081.75	31.03	1264961.75	31.32
	15	11	4	4,299	8,694.90	10,954.00	25.98	15,556.50	78.92	24684.5	183.9	10954	25.98
	16	11	5	748	28,908.00	28,908.00	0	28,908.00	0	78821.5	172.7	28908	0
Large	17	15	12	4,429	68,363.38	70,390.64	2.97	67,550.29	-1.19	69581.7875	1.78	67666.7875	-1.02
	18	16	6	1,121	31,768.90	39,751.61	25.13	33,475.93	5.37	91780.925	188.9	33475.925	5.37
	19	17	27	10,611	451,778.80	371,064.54	-17.87	510,624.00	13.03	914136.603	102.3	511298.403	13.17
	20	26	15	3,790	149,754.01	141,894.22	-5.25	134,174.42	-10.4	178039.66	18.89	153178.16	2.29

Chapter 6
Conclusions and Recommendations

6.1. Conclusion

In this thesis, a set of rectangular panels in small size is required to cut from a set of rectangular sheet in big size. Paper, metal bars, sheets, hardboards, leather, and cloth are needed to cut. Only Two-Dimensional Rectangular Guillotine Cutting are taken into account. Eight different techniques namely: 2DHC, 2DVC, 2DHI, 2DVI, SW_PH_HC, SW_PW_VC, MinSW_OPH_HC, and MinSH_OPW_VC techniques are proposed:

Firstly, 2D horizontal construction technique. A sheet set and a panel set are sorted based on height from the highest to the shortest one. Then each panel is picked to put into the sheet horizontally. Longitude is applied to each created pattern.

Secondly, 2D vertical construction technique. A sheet set is sorted in descending order based on height, and a panel set is sorted in descending order based on width. Then each panel is selected to put into each sheet vertically. Latitude cut is applied to each created pattern.

Thirdly, 2D horizontal improvement. It come from a combination of 2D simple heuristic cutting and 2D horizontal construction. A panel set is sorted based on height from the highest to the shortest one. Each panel is picked to put horizontally into each sheet type until no more space to fill in. A pattern with a minimum waste to selected to cut horizontally. This process is needed to do repeatedly until all the demand is fulfilled.

Fourthly, 2D vertical improvement. It come from a combination of 2D simple heuristic cutting and 2D vertical construction. A panel set is sorted based on width from the highest to the shortest one. Each panel is picked to put vertically into each sheet type until no more space to fill in. A pattern with a minimum waste to selected to cut vertically. This process is needed to do repeatedly until all the demand is fulfilled.

Fifthly, Sheet Width Panel Height Horizontal Cut technique. A sheet set is sorted based on width, and a panel set is sorted based on height from the highest to the shortest one. Then each panel is selected to put into the sheet horizontally. Longitude cut is applied to each created patterns.

Sixthly, Sheet Width Panel Width Vertical Cut technique. A sheet set is sorted based on width, and a panel set is also sorted based on width from the highest to the shortest one. Then each panel is selected to put into the sheet vertically. Latitude cut is applied to each created patterns.

Seventhly, Minimum Sheet Width Ordering Panel Height Horizontal Cut technique. A panel set is sorted in accordance with height. The first panels is selected to put in a longitude way into each sheet type in a sheet set to cut horizontally. Leftover width of each sheet is calculated to divide with each panel width. The panel that given the smallest remainder is picked to put after the first selected panel. This process needed to do repeatedly. Only the pattern that provide a minimum total waste is kept to cut.

Eighthly, Minimum Sheet Height Ordering Panel Width Vertical cut. A panel set is sorted in accordance with width. Then the first panels is selected to put in a latitude way into each sheet type in a sheet set to cut vertically. Leftover height of each sheet is calculated to divide with each panel width. The panel that given the smallest remainder is selected to put on the first selected panel vertically. This process needs to do repeatedly. Only the pattern that provide a minimum total waste is kept to cut.

The output of these eight proposed techniques are evaluated by comparing with Column Generation, which is used as a benchmark, and 2D Simple Heuristic Cutting, a simple cutting method, in term of both total waste and computational time.

Twenty different size of instances are tested with these 10 techniques. The result indicates that all proposed techniques can provide a better patterns in some instances compared to 2D simple heuristic cutting. Only 4 out of 8 proposed techniques can provide some good pattern comparing to column generation. They are 2D horizontal improvement, 2D vertical improvement, Minimum Sheet Width Ordering Panel Height Horizontal Cut method, and the Minimum Sheet Height Ordering Panel Width Vertical cut. These 4 techniques are not overlapped each other. It totally depends on the size of sheet in a sheet set and the size of panel in the panel set. It would be great

if we combine these four techniques together. Only the best pattern from each technique of each instance is selected to cut.

All proposed techniques can provide the pattern of cutting in a very short computational time. The computational time increases exponentially for column generation technique when the size of the instance is bigger and bigger.

6.2. Recommendation for Further Study

For further study, 2D 3stage can be taken into consideration. In additional, this problem should be applied by using meta-heuristic technique as well. Some scholars suggested meta-heuristics as in the following. They are Harmony Search Algorithm (HSA) proposed by Zong Woo Geem et al. (2001), Genetic Algorithms (GAs) proposed by Holland (1975), Simulated Annealing (SA) proposed by Kirkpatrick and Vecchi (1983), Particle Swarm Optimization (PSO) proposed by Kennedy and Eberhart (1995), Bee Algorithms (BA) proposed by Pham et al. (2006), Ant System (AS) proposed by Drigo et al. (1996), or Tabu Search (TS) proposed by Glover (1977) to help exploring further good pattern.

References

Alvarez-Valdes, R., Parajon, A., & Tamarit, J. M. (2002). A computational study of LP-based heuristic algorithms for two-dimensional guillotine cutting stock problems. *OR Spectrum, 24*(2), 179-192. doi: 10.1007/s00291-002-0093-3

Andrade, R., Birgin, E. G., & Morabito, R. (2013). Two-stage two-dimensional guillotine cutting problems with usable leftovers. *Department of Computer Science, Institute of Mathematics and Statistics, University of Sao Paulo, Brazil.*

Ayachi.I, Kammarti.R, Ksouri.M, & Borne.P. (2010). Harmony Search Algorithm for the Container Storage Problem. *8th International Conference of Modeling and Simulation*.

Cerqueira, G. R. L., & Yanasse, H. H. (2009). A pattern reduction procedure in a one-dimensional cutting stock problem by grouping items according to their demands. *Journal of Computational Interdisciplinary Sciences, 1(2)*, 159-164.

Cui, Y. (2004). Generating optimal T-shape cutting patterns for rectangular blanks. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, 218*(8), 857-866.

Cui, Y. (2012a). A CAM system for one-dimensional stock cutting. *Advances in Engineering Software, 47*(1), 7-16. doi: 10.1016/j.advengsoft.2011.12.004

Cui, Y. (2012b). Fast heuristic for constrained homogenous T-shape cutting patterns. *Applied Mathematical Modelling, 36*(8), 3696-3711. doi: 10.1016/j.apm.2011.11.005

Cui, Y. (2012c). A new dynamic programming procedure for three-staged cutting patterns. *Journal of Global Optimization, 55*(2), 349-357. doi: 10.1007/s10898-012-9930-3

Cui, Y., & Huang, B. (2012). Heuristic for constrained T-shape cutting patterns of rectangular pieces. *Computers & Operations Research, 39*(12), 3031-3039. doi: 10.1016/j.cor.2012.03.001

Cui, Y., & Liu, Z. (2007). T-shape homogenous block patterns for the two-dimensional cutting problem. *Journal of Global Optimization, 41*(2), 267-281. doi: 10.1007/s10898-007-9252-z

Cui, Y., Yang, L., Zhao, Z., Tang, T., & Yin, M. (2013). Sequential grouping heuristic for the two-dimensional cutting stock problem with pattern reduction. *International Journal of Production Economics, 144*(2), 432-439. doi: 10.1016/j.ijpe.2013.03.011

Cui, Y., & Zhao, Z. (2013). Heuristic for the rectangular two-dimensional single stock size cutting stock problem with two-staged patterns. *European Journal of Operational Research, 231*(2), 288-298. doi: 10.1016/j.ejor.2013.05.042

Drigo, M., Maniezzo, V., & Colorni, A. (1996). The ant system: optimization by a colony of cooperation agents. *IEEE Transactions of Systems, Man, and Cybernetics*(Part B), 29-41.

Geem, Z. W. (2006). *Improved harmony search from ensemble of music players.* Paper presented at the Knowledge-Based Intelligent Information and Engineering Systems.

Geem, Z. W. (2008). Harmony Search Applications in Industry. In B. Prasad (Ed.), *Soft Computing Applications in Industry* (Vol. 226, pp. 117-134): Springer Berlin Heidelberg.

Geem, Z. W., Kim, J. H., & Loganathan, G. (2001). A new heuristic optimization algorithm: harmony search. *Simulation, 76*(2), 60-68.

Geem, Z. W., Lee, K. S., & Park, Y. (2005). Application of harmony search to vehicle routing. *American Journal of Applied Sciences, 2*(12), 1552.

Gilmore, P. C., & Gomory, R. E. (1965). Multistage cutting stock problems of two and more dimensions. *Operation Research, 13*, 94-120.

Glover, F. (1977). Heuristics for integer programming using surrogate constraints. *Decision Sciences, 8*(1), 156-166.

Holland, J. (1975). Adaptation in natural and artificial systems. University of Michigan Press. *Ann Arbor, 1*(975), 1.

Kennedy, J., & Eberhart, R. (1995, Nov/Dec 1995). *Particle swarm optimization.* Paper presented at the Neural Networks, 1995. Proceedings., IEEE International Conference on.

Kirkpatrick, S., & Vecchi, M. (1983). Optimization by simmulated annealing. *science, 220*(4598), 671-680.

Lee, K. S., Geem, Z. W., Lee, S.-h., & Bae, K.-w. (2005). The harmony search heuristic algorithm for discrete structural optimization. *Engineering Optimization, 37*(7), 663-684.

Lodi, A., & Monaci, M. (2003). Integer linear programming models for 2-staged two-dimensional Knapsack problems. *Mathematical Programming, 94*(2-3), 257-278. doi: 10.1007/s10107-002-0319-9

Pham, D., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S., & Zaidi, M. (2006). *The bees algorithm-a novel tool for complex optimisation problems.* Paper presented at the Proceedings of the 2nd virtual international conference on intelligent production machines and systems (IPROMS 2006).

Pichpibul, T., & Kawtummachai, R. (2013). *Modified Harmony Search Algorithm for the Capacitated Vehicle Routing Problem.* Paper presented at the Proceedings of the International Multi Conference of Engineers and Computer Scientists.

Suliman, S. M. A. (2006). A sequential heuristic procedure for the two-dimensional cutting-stock problem. *International Journal of Production Economics, 99*(1-2), 177-185. doi: DOI 10.1016/j.ijpe.2004.12.017

Worasucheep, C. (2011). A harmony search with adaptive pitch adjustment for continuous optimization. *International Journal of Hybrid Information Technology, 4*(4).

Yanasse, H. H., & Limeira, M. S. (2006). A hybrid heuristic to reduce the number of different patterns in cutting stock problems. *Computers & Operations Research, 33*(9), 2744-2756. doi: 10.1016/j.cor.2005.02.026

Yanasse, H. H., & Morabito, R. (2008). A note on linear models for two-group and three-group two-dimensional guillotine cutting problems. *International Journal of Production Research, 46*(21), 6189-6206. doi: Doi 10.1080/00207540601011543

Yanasse, H. H., Zinober, A. S. I., & HARRIS, R. G. (1991). Two-dimensional Cutting Stock with Multiple Stock Sizes. *J. Opl Res. Soc., 42*(8), 673-683.

Appendices

7.1. Appendix A: Java Source Code

7.1.1. 2D Simple Heuristic Cutting

```
package Simple_Heuristic_II;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class KCE_Simple_Heuristic_Cutting{
  public static void main(String[] args) {
    long startTime = System.currentTimeMillis();
    ArrayList<Double> sWidth = new ArrayList<Double>();
    ArrayList<Double> sHeight = new ArrayList<Double>();
    ArrayList<Double> nSheet = new ArrayList<Double>();
    ArrayList<Double> pWidth = new ArrayList<Double>();
    ArrayList<Double> pHeight = new ArrayList<Double>();
    ArrayList<Double> nPanel = new ArrayList<Double>();
    ArrayList<Double> sLeftOverArea = new ArrayList<Double>();
    String excelFile = "D:\\Input and output of heuristics based on height_KCE.xlsx";

    System.out.println("ArrayList in row and column set: ");
    ReadExcel(sWidth, sHeight, nSheet, pWidth, pHeight, nPanel, excelFile);

    ArrayList<Double> sheetWidth = new ArrayList<Double>();
    ArrayList<Double> sheetHeight = new ArrayList<Double>();
    ArrayList<Double> panelWidth = new ArrayList<Double>();
    ArrayList<Double> panelHeight = new ArrayList<Double>();
    ArrayList<Double> overCutPanel = new ArrayList<Double>();
    ArrayList<Double> overUnderCutPanel = new ArrayList<Double>();
    ArrayList<Double> wasteForEachOverCutPanelForEachSheet = new
ArrayList<Double>();
    ArrayList<Double> totalWastePerSheetArray = new ArrayList<Double>();

    //define sheet set
    System.out.println("\n" + "Sheet set: ");
```

```
    for (int i=0; i<nSheet.size(); i++) {
      for (int a=0; a<nSheet.get(i); a++) {
        sheetWidth.add(sWidth.get(i));
        sheetHeight.add(sHeight.get(i));
        sLeftOverArea.add(sWidth.get(i) * sHeight.get(i));
        System.out.print( "[" +sHeight.get(i)+" x "+sWidth.get(i)+"]" + ", ");
      }System.out.println();
    }
    //define panel set
    System.out.println("\n" + "Panel set:");
    for (int i=0; i<nPanel.size(); i++) {
      for (int a=0; a<nPanel.get(i); a++) {
        panelWidth.add(pWidth.get(i));
        panelHeight.add(pHeight.get(i));
        System.out.print( "["+pHeight.get(i)+" x "+pWidth.get(i)+ "]" + ", ");
      }System.out.println();
    }
    // To check and remove the big panel that we cannot put into the sheet, if we don't
have it, it might cause to error.
    double maxSHeight = 0;
    double maxSWidth = 0;
    for (int j = 0; j < nPanel.size(); j++) {
      for (int i = 0; i < nSheet.size(); i++) {
        if (sHeight.get(i) > maxSHeight) {
          maxSHeight = sHeight.get(i);
        }
        if (sWidth.get(i) > maxSWidth) {
          maxSWidth = sWidth.get(i);
        }
      }
      if (((pHeight.get(j) > maxSHeight)) || (pWidth.get(j) > maxSWidth)) {
        pWidth.remove(j);
        pHeight.remove(j);
        nPanel.remove(j);
        j--;
      }
    }
    //To check and remove the small sheet that we cannot use to cut other panel, if we
don't have it, it might cause to error
    double minPHeight = 999999999;
    double minPWidth = 999999999;
    for (int i = 0; i < nSheet.size(); i++) {
      for (int j = 0; j < nPanel.size(); j++) {
        if (minPHeight > pHeight.get(j)) {
          minPHeight = pHeight.get(j);
        }
        if (minPWidth > pWidth.get(j)) {
```

```
            minPWidth = pWidth.get(j);
          }
        }
        if ((sHeight.get(i) < minPHeight) || (sWidth.get(i) < minPWidth)) {
          sWidth.remove(i);
          sHeight.remove(i);
          nSheet.remove(i);
          i--;
        }
      }
      //here is the process of cutting
      try {
        FileInputStream file = new FileInputStream(new File(excelFile));
        XSSFWorkbook workbook = new XSSFWorkbook(file);
        XSSFSheet sheet = workbook.getSheet("Output");
        Row row; // Declare row variable as Excel row.
        Cell cell;  // Declare cell variable as Excel cell.

        System.out.println("\n" + "Here is the step of cutting: ");
        ArrayList<Integer> chosenPanelIndex = new ArrayList<Integer>();
        ArrayList<Integer> chosenPanelAmount = new ArrayList<Integer>();

        ArrayList<Integer> usedPanelIndex = new ArrayList<Integer>();//the index of
the panel that have been cut in each type of sheet
        ArrayList<Integer> usedPanelAmount = new ArrayList<Integer>();//number of
panel in each type that used to cut
        ArrayList<Double> leftOverDemand = new ArrayList<Double>();//number of
demand after update with the panel number cut
        ArrayList<Double> xOrdinateArray = new ArrayList<Double>();//Use to add the
x ordinate one by one then when the panel is cut then clone it to xOrdinateSelected
when the sheet is full
        ArrayList<Double> yOrdinateArray = new ArrayList<Double>();
        ArrayList<Double> xOrdinateSelected = new ArrayList<Double>();//use to write
the coordinate in excel
        ArrayList<Double> yOrdinateSelected = new ArrayList<Double>();
        ArrayList<Double> pHeightArray = new ArrayList<Double>();//Use to add the
panel cut in order one by one then clone it to pHeightSelected when the sheet cut is
full
        ArrayList<Double> pWidthArray = new ArrayList<Double>();
        ArrayList<Double> pHeightSelected = new ArrayList<Double>();//use to write
the panel cut in excel in accordance with the coordinate
        ArrayList<Double> pWidthSelected = new ArrayList<Double>();

        ArrayList<Double> panelHeightAtBeginningLevel = new ArrayList<Double>();
        ArrayList<Double> panelHeightAtBeginningLevelClone = new
ArrayList<Double>();
```

```
        double leftoverHeight;
        double leftoverWidth;
        boolean existingLevel = false;// New or existing level check
        double currentLeftOverArea=0;
        double smallestLeftOverArea=0;// Variable to store the smallest left over area of
sheet.
        int smallestLeftOverSheet=0;     // Store index of the smallest left over area sheet.
        int numberPanelPerSheet=0;
        int panelAmountWithMinimumWaste=0;
        double sheetNumberNeeded;
        long endTime = 0;
        double xOrdinate = 0;
        double yOrdinate = 0;
        double w = 0;
        int startRow = 2, numberingCell = 0, sHeightCell = 1, sWidthCell = 2,
pHeightCell = 3, pWidthCell = 4, demandCell = 5, wastePerSheetCell = 6,
numberPanelPerSheetCell = 7, numberSheetNeededToCut = 8, leftOverPanelCell = 9,
computeTimeCell = 10, xCell = 11, yCell = 12,        overCutPanelCell = 13,
totalWastePerSheetCell = 14, wasteForCuttingCell = 15;
        int r=0;
        int printNo = 0;
        int i=0, j=0, k=0, level = 0;
        int count = 0;
        double totalWastePerSheet = 0;
        //this loop for pick sheet one by one from all sheet
        for (j = 0; j < nPanel.size(); j++) {
          smallestLeftOverArea = sWidth.get(0)*sHeight.get(0);
          row = sheet.createRow(startRow + r);
          for  (i = 0; i < sHeight.size(); i++) {
            numberPanelPerSheet = 0;
            leftoverHeight = sHeight.get(i);
            level = 0;
            System.out.print("\n" + "Sheet: " + (i+1) + " Size: " + sWidth.get(i) + " x " +
sHeight.get(i));
            currentLeftOverArea =  sWidth.get(i) * sHeight.get(i);
            //this loop use to define each level in a specific sheet
            for (double h=leftoverHeight; h>=pHeight.get(pHeight.size()-1);
h=leftoverHeight) {
              leftoverWidth = sWidth.get(i);
              existingLevel = false;
              ArrayList<Double>newPWidth = new ArrayList<Double>();
              ArrayList<Double>newPHeight = new ArrayList<Double>();
              ArrayList<Double>newNPanel = new ArrayList<Double>();
              if (sHeight.get(i) == leftoverHeight) {//use to start the coordinate of y
                yOrdinate = 0;
              }
```

```
            panelHeightAtBeginningLevel.add(pHeight.get(0));// Just want to add the
first panel before other panel add in
            for (int l = 0; l < pHeight.size(); l++) {// use to continue to the next level by
increase y
              if (sHeight.get(i)!= leftoverHeight) {
                if (pHeight.get(l) <= leftoverHeight) {
                  yOrdinate = yOrdinate + panelHeightAtBeginningLevelClone.get(1);

                  break;
                }
                if (pHeight.get(l) > leftoverHeight){
                  continue;
                }
              }
            }
            level += 1;
            System.out.println();
            System.out.print("\tLevel: " + level + " |");
            //to check the panel in case that it satisfies only one criteria like sample 34

            if (existingLevel == false && ((leftoverHeight>=pHeight.get(j) &&
leftoverWidth>=pWidth.get(j)) )){
              existingLevel = true;
            } else {
              newPWidth = (ArrayList)pWidth.clone();
              newPHeight =(ArrayList)pHeight.clone();
              newNPanel = (ArrayList)nPanel.clone();
              newPWidth.remove(pWidth.size()-1);
              newPHeight.remove(pHeight.size()-1);
              newNPanel.remove(nPanel.size()-1);
            }
            newPWidth.clear();
            newPHeight.clear();
            newNPanel.clear();
            panelHeightAtBeginningLevelClone.clear();
            //Start cutting for the first panel that put into the sheet
            if (leftoverHeight >= pHeight.get(j)) {
              for ( w = leftoverWidth; w >= pWidth.get(j); w = leftoverWidth) {        //
To do: make this loop take demand into consideration.
                if (existingLevel == true && leftoverWidth>=pWidth.get(j)) {
                  System.out.print("  " + pWidth.get(j) + " x " + pHeight.get(j) + "  |");
                  if (sWidth.get(i) == leftoverWidth) {
                    pHeightArray.add(pHeight.get(j));
                    pWidthArray.add(pWidth.get(j));
                    panelHeightAtBeginningLevel.add(pHeight.get(j));
                    xOrdinate = 0;
                    xOrdinateArray.add(xOrdinate);
```

```
                    yOrdinateArray.add(yOrdinate);
                  } else {
                    pHeightArray.add(pHeight.get(j));
                    pWidthArray.add(pWidth.get(j));
                    xOrdinate = xOrdinate + pWidthArray.get(pWidthArray.size()-2);
                    xOrdinateArray.add(xOrdinate);
                    yOrdinateArray.add(yOrdinate);
                  }
                  leftoverWidth = leftoverWidth - pWidth.get(j);
                  currentLeftOverArea = currentLeftOverArea-
(pWidth.get(j)*pHeight.get(j));
                  numberPanelPerSheet += 1;
                  if(usedPanelIndex.indexOf(j) == -1) {
                    usedPanelIndex.add(j);
                    usedPanelAmount.add(1);
                  } else {
                    usedPanelAmount.set(usedPanelIndex.indexOf(j),
usedPanelAmount.get(usedPanelIndex.indexOf(j))+1);//use to increase the amount of
the same panel size that put into the sheet
                  }
                }
              }
            }
            //Start cutting for the next panel that put into the sheet when we put other
types of panel in the same level or the next level
            ArrayList<Double> nextPanelCut = new ArrayList<Double>();
            panelHeightAtBeginningLevelClone =
(ArrayList)panelHeightAtBeginningLevel.clone();
            panelHeightAtBeginningLevel.clear();
            leftoverHeight = leftoverHeight - pHeight.get(j);
          }
          System.out.println();
          System.out.print("Total Waste for each sheet: " + currentLeftOverArea);
          System.out.print("\nPanel fit: " + numberPanelPerSheet);
          System.out.println();
          if (smallestLeftOverArea > currentLeftOverArea) {
            smallestLeftOverArea = currentLeftOverArea;
            smallestLeftOverSheet = i;
            panelAmountWithMinimumWaste = numberPanelPerSheet;
            pHeightSelected = (ArrayList)pHeightArray.clone();
            pWidthSelected = (ArrayList)pWidthArray.clone();

            xOrdinateSelected = (ArrayList)xOrdinateArray.clone();
            yOrdinateSelected = (ArrayList)yOrdinateArray.clone();

            chosenPanelIndex = (ArrayList)usedPanelIndex.clone();
```

```
            chosenPanelAmount = (ArrayList)usedPanelAmount.clone();
          }
          for(int ind =0; ind < chosenPanelIndex.size(); ind++) {
            System.out.println("Used Index " + chosenPanelIndex.get(ind));
            System.out.println("Used Amount " + chosenPanelAmount.get(ind));
          }
          // clear it first before selecting the next panel to cut; otherwise, it adds on the
existing data
          pHeightArray.clear();
          pWidthArray.clear();

          xOrdinateArray.clear();
          yOrdinateArray.clear();

          usedPanelIndex.clear();
          usedPanelAmount.clear();
        }
        System.out.println();
        System.out.println(panelAmountWithMinimumWaste + " panels cut from panel
set by putting in sheet number " + (smallestLeftOverSheet+1) + " (" +
sWidth.get(smallestLeftOverSheet) + " x " + sHeight.get(smallestLeftOverSheet) + ")
with the minimum waste " + smallestLeftOverArea);

        double minimumSheet =
10000000;//Math.ceil(nPanel.get(chosenPanelIndex.get(0)) /
chosenPanelAmount.get(0))
        double result;
        // This loop use to select the pattern that can give the minimum waste to cut.
        for(int d=0; d < chosenPanelIndex.size(); d++) {
          result = Math.ceil(nPanel.get(chosenPanelIndex.get(d)) /
chosenPanelAmount.get(d));
          if (minimumSheet > result) {
            minimumSheet = result;
          }
          System.out.println("To fulfill demand of " +
nPanel.get(chosenPanelIndex.get(d)) + " we need to use " + result + " sheets");
        }
        System.out.println("\nTherefore: To satisfy the demand we need: " +
minimumSheet + " sheets.");
        //To find the over cut panel
        double totalWasteOfOverCutPanel = 0;
        for (int l = 0; l < chosenPanelIndex.size(); l++) {
          overUnderCutPanel.add((minimumSheet * chosenPanelAmount.get(l)) -
nPanel.get(chosenPanelIndex.get(l)));
          if (overUnderCutPanel.get(l)<= 0) {
            overCutPanel.add(0.0);
          } else {
```

```
            overCutPanel.add(overUnderCutPanel.get(l));
          }
          wasteForEachOverCutPanelForEachSheet.add(overCutPanel.get(l)*
pHeight.get(l) * pWidth.get(l));
          totalWasteOfOverCutPanel = totalWasteOfOverCutPanel +
wasteForEachOverCutPanelForEachSheet.get(l);//To find the waste of the over cut
panel
          System.out.println("Over cut Panel = " + overCutPanel);
        }
        totalWastePerSheet = (smallestLeftOverArea * minimumSheet) +
totalWasteOfOverCutPanel;//Total waste per sheet included the over cut panel
        totalWastePerSheetArray.add(totalWastePerSheet);

        int index;
        double amount;
        double demand;
        //Writing the panel of the pattern that we have selected to cut
        for(int ind = 0; ind < chosenPanelIndex.size(); ind++) {
          System.out.println("Chosen Index " + chosenPanelIndex.get(ind));
          System.out.println("Used Amount " + chosenPanelAmount.get(ind));
          row = sheet.createRow(startRow + r);
          index = chosenPanelIndex.get(ind);
          amount = chosenPanelAmount.get(ind);
          demand = nPanel.get(chosenPanelIndex.get(ind));
          leftOverDemand.add(demand - (amount * minimumSheet));
          count = 0;
          //Write only that first stage of cutting like No
          if (ind == 0) {//ind = chosenPanelIndex
            cell = row.createCell(numberingCell);
            cell.setCellValue(printNo+1);

            cell = row.createCell(sHeightCell);
            cell.setCellValue(sHeight.get(smallestLeftOverSheet));

            cell = row.createCell(sWidthCell);
            cell.setCellValue(sWidth.get(smallestLeftOverSheet));

          if (leftOverDemand.get(ind)> 0) {
            cell = row.createCell(leftOverPanelCell);
            cell.setCellValue(leftOverDemand.get(ind));
          }
        }
        if (ind < chosenPanelIndex.size()-1) { //do not let it increase the row when it is
the last panel that we put coz we have to put other data on the last row
          cell = row.createCell(demandCell);
          cell.setCellValue(nPanel.get(chosenPanelIndex.get(ind)));
```

```
cell = row.createCell(overCutPanelCell);
cell.setCellValue(overCutPanel.get(ind));

cell = row.createCell(numberPanelPerSheetCell);
cell.setCellValue(chosenPanelAmount.get(ind));
if (leftOverDemand.get(ind)> 0) {
  cell = row.createCell(leftOverPanelCell);
  cell.setCellValue(leftOverDemand.get(ind));
}
// to spread the panel of cutting such that i can put the coordinate
for (int l = 0; l < chosenPanelAmount.get(ind); l++) {

  cell = row.createCell(pHeightCell);
  cell.setCellValue(pHeightSelected.get(l));

  cell = row.createCell(pWidthCell);
  cell.setCellValue(pWidthSelected.get(l));

  cell = row.createCell(xCell);
  cell.setCellValue(xOrdinateSelected.get(l));

  cell = row.createCell(yCell);
  cell.setCellValue(yOrdinateSelected.get(l));

  count = count + 1;
  r++;
  row = sheet.createRow(startRow + r);
 }
 //Remove that panel use such that it can print of other panel
 for (int b = (count-1) ; b >= 0; b--) {
  pHeightSelected.remove(b);
  pWidthSelected.remove(b);
  xOrdinateSelected.remove(b);
  yOrdinateSelected.remove(b);
 }
} else if ((chosenPanelIndex.size()-1) == ind) {
 cell = row.createCell(demandCell);
 cell.setCellValue(nPanel.get(chosenPanelIndex.get(ind)));

 cell = row.createCell(overCutPanelCell);
 cell.setCellValue(overCutPanel.get(ind));

 cell = row.createCell(numberPanelPerSheetCell);
 cell.setCellValue(chosenPanelAmount.get(ind));

 if (leftOverDemand.get(ind)> 0) {
  cell = row.createCell(leftOverPanelCell);
```

```
            cell.setCellValue(leftOverDemand.get(ind));
          }
          for (int l = 0; l < chosenPanelAmount.get(ind); l++) {
            cell = row.createCell(pHeightCell);
            cell.setCellValue(pHeightSelected.get(l));
            cell = row.createCell(pWidthCell);
            cell.setCellValue(pWidthSelected.get(l));
            cell = row.createCell(xCell);
            cell.setCellValue(xOrdinateSelected.get(l));
            cell = row.createCell(yCell);
            cell.setCellValue(yOrdinateSelected.get(l));
            count = count + 1;
            r++;
            row = sheet.createRow(startRow + r);
          }
          for (int b = (count-1) ; b >= 0; b--) {
            pHeightSelected.remove(b);
            pWidthSelected.remove(b);

            xOrdinateSelected.remove(b);
            yOrdinateSelected.remove(b);
          }
          r--;
        }
      }
      wasteForEachOverCutPanelForEachSheet.clear();
      overUnderCutPanel.clear();
      overCutPanel.clear();
      r = r + 2;
      printNo++;
      cell = row.createCell(wastePerSheetCell);
      cell.setCellValue(smallestLeftOverArea);

      cell = row.createCell(numberSheetNeededToCut);
      cell.setCellValue(minimumSheet);
      cell = row.createCell(totalWastePerSheetCell);
      cell.setCellValue(totalWastePerSheet);
      endTime = System.currentTimeMillis();
      System.out.println("Time taken for this process are: " + (endTime - startTime) +
" milli seconds" );
      cell = row.createCell(computeTimeCell);
      cell.setCellValue(endTime - startTime);
      for( int d = chosenPanelIndex.size()-1; d >= 0; d--) {
        index = chosenPanelIndex.indexOf(Collections.max(chosenPanelIndex));
        amount = chosenPanelAmount.get(index);
        demand = nPanel.get(chosenPanelIndex.get(index));
        int newIndex = chosenPanelIndex.get(index);
```

```
        if (leftOverDemand.get(index) > 0) {
          nPanel.set(newIndex, leftOverDemand.get(index));
          chosenPanelIndex.remove(index);
          chosenPanelAmount.remove(index);
          leftOverDemand.remove(index);
        } else {
          nPanel.remove(newIndex);
          pHeight.remove(newIndex);
          pWidth.remove(newIndex);
          chosenPanelIndex.remove(index);
          chosenPanelAmount.remove(index);
          leftOverDemand.remove(index);
        }
      }
      leftOverDemand.clear();
      System.out.println("----------------------------------------");
      j--;
      double wasteForCutting = 0;
      for (int l = 0; l < totalWastePerSheetArray.size(); l++) {
        wasteForCutting = wasteForCutting + totalWastePerSheetArray.get(l);
      }
      cell = row.createCell(wasteForCuttingCell);
      cell.setCellValue(wasteForCutting);
    }
    chosenPanelIndex.clear();
    chosenPanelAmount.clear();
    file.close();

    FileOutputStream outFile =new FileOutputStream(new File(excelFile));
    workbook.write(outFile);
    outFile.close();
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
  public static void ReadExcel(ArrayList<Double> sWidth,
    ArrayList<Double> sHeight, ArrayList<Double> nSheet,
    ArrayList<Double> pWidth, ArrayList<Double> pHeight,
    ArrayList<Double> nPanel, String excelFile) {
    try {
      FileInputStream file = new FileInputStream(new File(excelFile));
      XSSFWorkbook workbook = new XSSFWorkbook(file);
      XSSFSheet sheet = workbook.getSheet("Input");
      // cRow = currentRow, pRow = processingRow
      int cRow = 0 , pRow = 0;
```

```
int cColumn = 1;
int dataInRow = 0;
//Iterate through each rows from first sheet
Iterator<Row> rowIterator = sheet.iterator();
while(rowIterator.hasNext()) {
  Row row = rowIterator.next();
  cRow++;
  cColumn = 1;
  dataInRow = 0;
  //For each row, iterate through each columns
  Iterator<Cell> cellIterator = row.cellIterator();
  while(cellIterator.hasNext()) {
    Cell cell = cellIterator.next();
    switch(cell.getCellType()) {
    case Cell.CELL_TYPE_BOOLEAN:
      System.out.print(cell.getBooleanCellValue() + "\t\t");
      dataInRow ++;
      break;
    case Cell.CELL_TYPE_NUMERIC:
      System.out.print(cell.getNumericCellValue() + "\t\t");
      if(cRow>pRow) {
        pRow = cRow;
    }
    else
    {
      switch(cColumn) {
      case 1:
        pHeight.add(cell.getNumericCellValue());
        break;
      case 2:
        pWidth.add(cell.getNumericCellValue());
        break;
      case 3:
        nPanel.add(cell.getNumericCellValue());
        break;
      case 4:
        sHeight.add(cell.getNumericCellValue());
        break;
      case 5:
        sWidth.add(cell.getNumericCellValue());
        break;
      case 6:
        nSheet.add(cell.getNumericCellValue());
      }
      cColumn++;
    }
    dataInRow++;
```

```
            break;
            case Cell.CELL_TYPE_STRING:
              System.out.print(cell.getStringCellValue() + "\t\t");
              dataInRow++;
              break;
            }
          }
          if(dataInRow > 0) {
            System.out.println("");
          }
        }
        file.close();
      } catch (FileNotFoundException e) {
        e.printStackTrace();
      } catch (IOException e) {
        e.printStackTrace();
      }
    }
}
```

7.1.2. 2D Horizontal Construction

```
package SHeightBased_PHeightBased_2D_2Stage_Horizontal_Cutting_P1;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class SHeightBased_PHeightBased_2D_2Stage_Horizontal_Cutting_P1{
  public static void main(String[] args) {
    long startTime = System.currentTimeMillis();
    ArrayList<Double> sWidth = new ArrayList<Double>();
    ArrayList<Double> sHeight = new ArrayList<Double>();
    ArrayList<Double> nSheet = new ArrayList<Double>();
    ArrayList<Double> pWidth = new ArrayList<Double>();
    ArrayList<Double> pHeight = new ArrayList<Double>();
    ArrayList<Double> nPanel = new ArrayList<Double>();
    ArrayList<Double> sLeftOverArea = new ArrayList<Double>();
    String excelFile = "D:\\Input and output of heuristics For construction 1 and
2.xlsx";
    System.out.println("ArrayList in row and column set: ");
    try {
      FileInputStream file = new FileInputStream(new File(excelFile));
      XSSFWorkbook workbook = new XSSFWorkbook(file);
      XSSFSheet sheet = workbook.getSheet("Input");
      int cRow = 0 , pRow = 0;
      int cColumn = 1;
      int dataInRow = 0;
      Iterator<Row> rowIterator = sheet.iterator();
      while(rowIterator.hasNext()) {
        Row row = rowIterator.next();
        cRow++;
        cColumn = 1;
        dataInRow = 0;
        Iterator<Cell> cellIterator = row.cellIterator();
        while(cellIterator.hasNext()) {
          Cell cell = cellIterator.next();
          switch(cell.getCellType()) {
```

```
        case Cell.CELL_TYPE_BOOLEAN:
          System.out.print(cell.getBooleanCellValue() + "\t\t");
          dataInRow ++;
          break;
        case Cell.CELL_TYPE_NUMERIC:
          System.out.print(cell.getNumericCellValue() + "\t\t");
          if(cRow>pRow) {
            pRow = cRow;
          }else{
            switch(cColumn) {
            case 1:
              pHeight.add(cell.getNumericCellValue());
              break;
            case 2:
              pWidth.add(cell.getNumericCellValue());
              break;
            case 3:
              nPanel.add(cell.getNumericCellValue());
              break;
            case 4:
              sHeight.add(cell.getNumericCellValue());
              break;
            case 5:
              sWidth.add(cell.getNumericCellValue());
              break;
            case 6:
              nSheet.add(cell.getNumericCellValue());
            }
            cColumn++;
          }
          dataInRow++;
          break;
        case Cell.CELL_TYPE_STRING:
          System.out.print(cell.getStringCellValue() + "\t\t");
          dataInRow++;
          break;
        }
      }
      if(dataInRow > 0) {
        System.out.println("");
      }
    }
    file.close();
  } catch (FileNotFoundException e) {
    e.printStackTrace();
  } catch (IOException e) {
    e.printStackTrace();
```

```
    }
    ArrayList<Double> sheetWidth = new ArrayList<Double>();
    ArrayList<Double> sheetHeight = new ArrayList<Double>();
    ArrayList<Double> panelWidth = new ArrayList<Double>();
    ArrayList<Double> panelHeight = new ArrayList<Double>();
    ArrayList<Double> totalWastePerSheet = new ArrayList<Double>();
    //define sheet set
    System.out.println("\n" + "Sheet set: ");
    for (int i=0; i<nSheet.size(); i++) {
    for (int a=0; a<nSheet.get(i); a++) {
      sheetWidth.add(sWidth.get(i));
      sheetHeight.add(sHeight.get(i));
      sLeftOverArea.add(sWidth.get(i) * sHeight.get(i));
      System.out.print( "[" + sHeight.get(i) + " x " + sWidth.get(i) + "]" + ", ");
    }System.out.println();
  }
  //define panel set
  System.out.println("\n" + "Panel set:");
  for (int i=0; i<nPanel.size(); i++) {
    for (int a=0; a<nPanel.get(i); a++) {
      panelWidth.add(pWidth.get(i));
      panelHeight.add(pHeight.get(i));
      System.out.print( "[" + pHeight.get(i) + " x " + pWidth.get(i)+ "]" + ", ");
    }System.out.println();
  }
  //here is the process of cutting
  try {
    FileInputStream file = new FileInputStream(new File(excelFile));
    XSSFWorkbook workbook = new XSSFWorkbook(file);
    XSSFSheet sheet = workbook.getSheet("Output");
    Row row;   // Declare row variable as Excel row.
    Cell cell;    // Declare cell variable as Excel cell.
    System.out.println("\n" + "Here is the step of cutting: ");
    double leftoverHeight;
    double leftoverWidth;
    double wasteBeforeCut = 0;
    int startRow = 2, sHeightCell = 0, sWidthCell = 1, pHeightCell = 2, pWidthCell =
3, levelCell = 4, sLeftOverAreaCell = 5, computeTimeCell = 6, totalWasteCell = 7;
    int r=0;
    long computeTime = 0;
    boolean existingLevel = false;  // Level check if new calculate new
[leftoverHeight] and [leftoverWidth].
    int i=0, j=0, level;
    //this loop for pick sheet one by one from all sheet
    for (i = 0; i < sheetHeight.size(); i++) {
      leftoverHeight = sheetHeight.get(i);
      leftoverWidth = sheetWidth.get(i);//
```

```
      level = 0;
      row = sheet.createRow(startRow + r);
      cell = row.createCell(sHeightCell);
      cell.setCellValue(sheetHeight.get(i));
      cell = row.createCell(sWidthCell);
      cell.setCellValue(sheetWidth.get(i));
      //To define the value of the waste
      if (i > 0) {
        sheet.getRow(startRow + (r-
1)).createCell(sLeftOverAreaCell).setCellValue(sLeftOverArea.get(i-1));
      }
      //this loop use to define each level in a specific sheet
      for (double h=leftoverHeight; h>=panelHeight.get(panelHeight.size()-1);
h=leftoverHeight) {
        leftoverWidth = sheetWidth.get(i);
        existingLevel = false;
        // this loop use to pick the panel to put inside each level in a specific sheet
        for (j = 0; j < panelHeight.size(); j++) {
          if (existingLevel == false && (leftoverHeight>=panelHeight.get(j) &&
leftoverWidth>=panelWidth.get(j))) {
            leftoverHeight = leftoverHeight - panelHeight.get(j);
            existingLevel = true;
            level += 1;
          }
          if (existingLevel == true && leftoverWidth >= panelWidth.get(j)) {
            System.out.println("put panel: " + panelHeight.get(j) + " x " +
panelWidth.get(j) + " into sheet: " + (i+1) + " level: " + level);
            sLeftOverArea.set(i, sLeftOverArea.get(i) - (panelHeight.get(j) *
panelWidth.get(j)));
            try {
              row = sheet.getRow(startRow + r);
              if (row == null) {
                row = sheet.createRow(startRow + r);
              }
            }
            catch (Exception e) {
              row = sheet.createRow(startRow + r);
            }
            row.createCell(pHeightCell).setCellValue(panelHeight.get(j));
            row.createCell(pWidthCell).setCellValue(panelWidth.get(j));
            row.createCell(levelCell).setCellValue(level);
            r = r + 1;       // move to new row.
            leftoverWidth = leftoverWidth - panelWidth.get(j);
            panelHeight.remove(j);
            panelWidth.remove(j);
            j = j - 1;
            }
```

```
          }
          if(panelHeight.isEmpty()){
            sheet.getRow(startRow + (r-
1)).createCell(sLeftOverAreaCell).setCellValue(sLeftOverArea.get(i));
            sheet.getRow(startRow + (r-
1)).createCell(computeTimeCell).setCellValue(computeTime);
            totalWastePerSheet.add(sLeftOverArea.get(i));
            for (int k = 0; k < totalWastePerSheet.size(); k++) {
              wasteBeforeCut = wasteBeforeCut + totalWastePerSheet.get(k);
            }
            sheet.getRow(startRow + (r-
1)).createCell(totalWasteCell).setCellValue(wasteBeforeCut);
            file.close();
            FileOutputStream outFile =new FileOutputStream(new File(excelFile));
            workbook.write(outFile);
            outFile.close();
            return;
          }
        }
        totalWastePerSheet.add(sLeftOverArea.get(i));
        sheetHeight.remove(0);
        sheetWidth.remove(0);
        long endTime = System.currentTimeMillis();
        computeTime = endTime - startTime;
        System.out.println("Time taken for this process are: " + (computeTime) + "
milli seconds" );
        cell = row.createCell(computeTimeCell);
        cell.setCellValue(computeTime);
      }
      sheet.getRow(startRow + (r-
1)).createCell(sLeftOverAreaCell).setCellValue(sLeftOverArea.get(9));
      file.close();
      FileOutputStream outFile =new FileOutputStream(new File(excelFile));
      workbook.write(outFile);
      outFile.close();
      for(int l=0; l < sLeftOverArea.size(); l++) {
        System.out.println("Sheet: " + (l+1) + " Left over area: " +
sLeftOverArea.get(l));
      }
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

7.1.3. 2D Vertical Construction

```
package SHeightBased_PWidthBased_2D_2Stage_Vertical_Cutting_P1;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class SHeightBased_PWidthBased_2D_2Stage_Vertical_Cutting_P1{
  public static void main(String[] args) {
    long startTime = System.currentTimeMillis();
    ArrayList<Double> sWidth = new ArrayList<Double>();
    ArrayList<Double> sHeight = new ArrayList<Double>();
    ArrayList<Double> nSheet = new ArrayList<Double>();
    ArrayList<Double> pWidth = new ArrayList<Double>();
    ArrayList<Double> pHeight = new ArrayList<Double>();
    ArrayList<Double> nPanel = new ArrayList<Double>();
    ArrayList<Double> sLeftOverArea = new ArrayList<Double>();
    String excelFile = "D:\\Input and output of heuristics For construction 1 and
2.xlsx";
    System.out.println("ArrayList in row and column set: ");
    try {
      FileInputStream file = new FileInputStream(new File(excelFile));
      XSSFWorkbook workbook = new XSSFWorkbook(file);
      XSSFSheet sheet = workbook.getSheet("Input2");
      int cRow = 0 , pRow = 0;
      int cColumn = 1;
      int dataInRow = 0;
      //Iterate through each rows from first sheet
      Iterator<Row> rowIterator = sheet.iterator();
      while(rowIterator.hasNext()) {
        Row row = rowIterator.next();
        cRow++;
        cColumn = 1;
        dataInRow = 0;
        //For each row, iterate through each columns
        Iterator<Cell> cellIterator = row.cellIterator();
        while(cellIterator.hasNext()) {
          Cell cell = cellIterator.next();
```

```
      switch(cell.getCellType()) {
      case Cell.CELL_TYPE_BOOLEAN:
        System.out.print(cell.getBooleanCellValue() + "\t\t");
        dataInRow ++;
        break;
      case Cell.CELL_TYPE_NUMERIC:
        System.out.print(cell.getNumericCellValue() + "\t\t");
        if(cRow>pRow) {
          pRow = cRow;
        }else{
          switch(cColumn) {
          case 1:
            pHeight.add(cell.getNumericCellValue());
            break;
          case 2:
            pWidth.add(cell.getNumericCellValue());
            break;
          case 3:
            nPanel.add(cell.getNumericCellValue());
            break;
          case 4:
            sHeight.add(cell.getNumericCellValue());
            break;
          case 5:
            sWidth.add(cell.getNumericCellValue());
            break;
          case 6:
            nSheet.add(cell.getNumericCellValue());
          }
          cColumn++;
        }
        dataInRow++;
        break;
      case Cell.CELL_TYPE_STRING:
        System.out.print(cell.getStringCellValue() + "\t\t");
        dataInRow++;
        break;
      }
    }
    if(dataInRow > 0) {
      System.out.println("");
    }
  }
  file.close();
} catch (FileNotFoundException e) {
  e.printStackTrace();
} catch (IOException e) {
```

```
      e.printStackTrace();
    }
    ArrayList<Double> sheetWidth = new ArrayList<Double>();
    ArrayList<Double> sheetHeight = new ArrayList<Double>();
    ArrayList<Double> panelWidth = new ArrayList<Double>();
    ArrayList<Double> panelHeight = new ArrayList<Double>();
    ArrayList<Double> totalWastePerSheet = new ArrayList<Double>();
    //define sheet set
    System.out.println("\n" + "Sheet set: ");
    for (int i=0; i<nSheet.size(); i++) {
      for (int a=0; a<nSheet.get(i); a++) {
        sheetWidth.add(sWidth.get(i));
        sheetHeight.add(sHeight.get(i));
        sLeftOverArea.add(sWidth.get(i) * sHeight.get(i));
        System.out.print( "[" + sHeight.get(i) + " x " + sWidth.get(i) + "]" + ", ");
      }System.out.println();
    }
    //define panel set
    System.out.println("\n" + "Panel set:");
    for (int i=0; i<nPanel.size(); i++) {
      for (int a=0; a<nPanel.get(i); a++) {
        panelWidth.add(pWidth.get(i));
        panelHeight.add(pHeight.get(i));
        System.out.print( "[" + pHeight.get(i) + " x " + pWidth.get(i)+ "]" + ", ");
      }System.out.println();
    }
    //here is the process of cutting
    try {
      // Open excel file.
      FileInputStream file = new FileInputStream(new File(excelFile));
      XSSFWorkbook workbook = new XSSFWorkbook(file);
      XSSFSheet sheet = workbook.getSheet("Output2");
      Row row; // Declare row variable as Excel row.
      Cell cell;  // Declare cell variable as Excel cell.
      System.out.println("\n" + "Here is the step of cutting: ");
      double leftoverHeight;
      double leftoverWidth;
      double wasteBeforeCut = 0;
      int startRow = 2, sHeightCell = 0, sWidthCell = 1, pHeightCell = 2, pWidthCell
= 3, levelCell = 4, sLeftOverAreaCell = 5, computeTimeCell = 6, totalWasteCell = 7;
      int r=0;
      long computeTime = 0;
      boolean existingLevel = false;
      int i=0, j=0, level;
      //this loop for pick sheet one by one from all sheet
      for (i = 0; i < sheetWidth.size(); i++) {
        leftoverHeight = sheetHeight.get(i);
```

```
        leftoverWidth = sheetWidth.get(i);
        level = 0;
        row = sheet.createRow(startRow + r);
        cell = row.createCell(sHeightCell);
        cell.setCellValue(sheetHeight.get(i));
        cell = row.createCell(sWidthCell);
        cell.setCellValue(sheetWidth.get(i));
        //To define the value of the waste
        if (i > 0) {
          sheet.getRow(startRow + (r-
1)).createCell(sLeftOverAreaCell).setCellValue(sLeftOverArea.get(i-1));
        }
        //this loop use to define each level in a specific sheet
        for (double h=leftoverWidth; h>=panelWidth.get(panelWidth.size()-1);
h=leftoverWidth) {
          leftoverHeight = sheetHeight.get(i);
          existingLevel = false;
          // this loop use to pick the panel to put inside each level in a specific sheet
          for (j = 0; j < panelWidth.size(); j++) {
            if (existingLevel == false && (leftoverHeight>=panelHeight.get(j) &&
leftoverWidth>=panelWidth.get(j))) {
              leftoverWidth = leftoverWidth - panelWidth.get(j);
              existingLevel = true;
              level += 1;
            }
            if (existingLevel == true && leftoverHeight>=panelHeight.get(j)) {
              System.out.println("put panel: " + panelHeight.get(j) + " x " +
panelWidth.get(j) + " into sheet: " + (i+1) + " level: " + level);
              sLeftOverArea.set(i, sLeftOverArea.get(i) - (panelHeight.get(j) *
panelWidth.get(j)));
              try {
                row = sheet.getRow(startRow + r);
                if (row == null) {
                  row = sheet.createRow(startRow + r);
                }
              }
              catch (Exception e) {
                row = sheet.createRow(startRow + r);
              }
              row.createCell(pHeightCell).setCellValue(panelHeight.get(j));
              row.createCell(pWidthCell).setCellValue(panelWidth.get(j));
              row.createCell(levelCell).setCellValue(level);
              r++;// move to new row.
              leftoverHeight = leftoverHeight - panelHeight.get(j);
              panelHeight.remove(j);
              panelWidth.remove(j);
              j = j - 1;
```

```
            }
          }
          if(panelHeight.isEmpty()){
            sheet.getRow(startRow + (r-
1)).createCell(sLeftOverAreaCell).setCellValue(sLeftOverArea.get(i));
            sheet.getRow(startRow + (r-
1)).createCell(computeTimeCell).setCellValue(computeTime);
            totalWastePerSheet.add(sLeftOverArea.get(i));
            for (int k = 0; k < totalWastePerSheet.size(); k++) {
              wasteBeforeCut = wasteBeforeCut + totalWastePerSheet.get(k);
            }
            sheet.getRow(startRow + (r-
1)).createCell(totalWasteCell).setCellValue(wasteBeforeCut);
            file.close();
            FileOutputStream outFile =new FileOutputStream(new File(excelFile));
            workbook.write(outFile);
            outFile.close();
            return;
          }
        }
        totalWastePerSheet.add(sLeftOverArea.get(i));
        sheetHeight.remove(0);
        sheetWidth.remove(0);
        long endTime = System.currentTimeMillis();
        computeTime = endTime - startTime;
        System.out.println("Time taken for this process are: " + (computeTime) + "
milli seconds" );
        cell = row.createCell(computeTimeCell);
        cell.setCellValue(computeTime);
      }
      sheet.getRow(startRow + (r-
1)).createCell(sLeftOverAreaCell).setCellValue(sLeftOverArea.get(9));
      file.close();
      FileOutputStream outFile =new FileOutputStream(new File(excelFile));
      workbook.write(outFile);
      outFile.close();
      for(int l=0; l < sLeftOverArea.size(); l++) {
        System.out.println("Sheet: " + (l+1) + " Left over area: " +
sLeftOverArea.get(l));
      }
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

7.1.4. 2D Horizontal Improvement

```
package Simple_Heuristic_II;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class Improve1_Simple_Heuristic_P_HeightBased_Cut_Horizontal{
  public static void main(String[] args) {
    long startTime = System.currentTimeMillis();
    ArrayList<Double> sWidth = new ArrayList<Double>();
    ArrayList<Double> sHeight = new ArrayList<Double>();
    ArrayList<Double> nSheet = new ArrayList<Double>();

    ArrayList<Double> pWidth = new ArrayList<Double>();
    ArrayList<Double> pHeight = new ArrayList<Double>();
    ArrayList<Double> nPanel = new ArrayList<Double>();

    ArrayList<Double> sLeftOverArea = new ArrayList<Double>();
    String excelFile = "D:\\Input and output of heuristics based on
height_ExactNumber.xlsx";

    System.out.println("ArrayList in row and column set: ");
    ReadExcel(sWidth, sHeight, nSheet, pWidth, pHeight, nPanel, excelFile);

    ArrayList<Double> sheetWidth = new ArrayList<Double>();
    ArrayList<Double> sheetHeight = new ArrayList<Double>();

    ArrayList<Double> panelWidth = new ArrayList<Double>();
    ArrayList<Double> panelHeight = new ArrayList<Double>();

    ArrayList<Double> overCutPanel = new ArrayList<Double>();
    ArrayList<Double> overUnderCutPanel = new ArrayList<Double>();

    ArrayList<Double> wasteForEachOverCutPanelForEachSheet = new
ArrayList<Double>();
```

```
    ArrayList<Double> totalWastePerSheetArray = new ArrayList<Double>();

    //define sheet set
    System.out.println("\n" + "Sheet set: ");
    for (int i=0; i<nSheet.size(); i++) {
      for (int a=0; a<nSheet.get(i); a++) {
        sheetWidth.add(sWidth.get(i));
        sheetHeight.add(sHeight.get(i));
        sLeftOverArea.add(sWidth.get(i) * sHeight.get(i));
        System.out.print( "[" + sHeight.get(i) + " x " + sWidth.get(i) + "]" + ", ");
      }System.out.println();
    }
    //define panel set
    System.out.println("\n" + "Panel set:");
    for (int i=0; i<nPanel.size(); i++) {
      for (int a=0; a<nPanel.get(i); a++) {
        panelWidth.add(pWidth.get(i));
        panelHeight.add(pHeight.get(i));
        System.out.print( "[" + pHeight.get(i) + " x " + pWidth.get(i)+ "]" + ", ");
      }System.out.println();
    }
    // To check and remove the big panel that we cannot put into the sheet, if we don't
have it, it might cause to error.
    double maxSHeight = 0;
    double maxSWidth = 0;
    for (int j = 0; j < nPanel.size(); j++) {
      for (int i = 0; i < nSheet.size(); i++) {
        if (sHeight.get(i) > maxSHeight) {
          maxSHeight = sHeight.get(i);
        }
        if (sWidth.get(i) > maxSWidth) {
          maxSWidth = sWidth.get(i);
        }
      }
      if (((pHeight.get(j) > maxSHeight)) || (pWidth.get(j) > maxSWidth)) {
        pWidth.remove(j);
        pHeight.remove(j);
        nPanel.remove(j);
        j--;
      }
    }
    //To check and remove the small sheet that we cannot use to cut other panel, if we
don't have it, it might cause to error
    double minPHeight = 999999999;
    double minPWidth = 999999999;
    for (int i = 0; i < nSheet.size(); i++) {
      for (int j = 0; j < nPanel.size(); j++) {
```

```
        if (minPHeight > pHeight.get(j)) {
          minPHeight = pHeight.get(j);
        }
        if (minPWidth > pWidth.get(j)) {
          minPWidth = pWidth.get(j);
        }
      }
      if ((sHeight.get(i) < minPHeight) || (sWidth.get(i) < minPWidth)) {
        sWidth.remove(i);
        sHeight.remove(i);
        nSheet.remove(i);
        i--;
      }
    }
    //here is the process of cutting
    try {
      FileInputStream file = new FileInputStream(new File(excelFile));
      XSSFWorkbook workbook = new XSSFWorkbook(file);
      XSSFSheet sheet = workbook.getSheet("Output");
      Row row;
      Cell cell;
      System.out.println("\n" + "Here is the step of cutting: ");

      ArrayList<Integer> chosenPanelIndex = new ArrayList<Integer>();
      ArrayList<Integer> chosenPanelAmount = new ArrayList<Integer>();

      ArrayList<Integer> usedPanelIndex = new ArrayList<Integer>();
      ArrayList<Integer> usedPanelAmount = new ArrayList<Integer>();
      ArrayList<Double> leftOverDemand = new ArrayList<Double>();

      ArrayList<Double> xOrdinateArray = new ArrayList<Double>();
      ArrayList<Double> yOrdinateArray = new ArrayList<Double>();

      ArrayList<Double> xOrdinateSelected = new ArrayList<Double>();
      ArrayList<Double> yOrdinateSelected = new ArrayList<Double>();

      ArrayList<Double> pHeightArray = new ArrayList<Double>();
      ArrayList<Double> pWidthArray = new ArrayList<Double>();

      ArrayList<Double> pHeightSelected = new ArrayList<Double>();
      ArrayList<Double> pWidthSelected = new ArrayList<Double>();

      ArrayList<Double> panelHeightAtBeginningLevel = new ArrayList<Double>();
      ArrayList<Double> panelHeightAtBeginningLevelClone = new
ArrayList<Double>();

      double leftoverHeight;
```

```
        double leftoverWidth;
        boolean existingLevel = false;
        double currentLeftOverArea=0;
        double smallestLeftOverArea=0;
        int smallestLeftOverSheet=0;
        int numberPanelPerSheet=0;
        int panelAmountWithMinimumWaste=0;
        double sheetNumberNeeded;
        long endTime = 0;
        double xOrdinate = 0;
        double yOrdinate = 0;
        double w = 0;
        int startRow = 2, numberingCell = 0, sHeightCell = 1, sWidthCell = 2,
pHeightCell = 3, pWidthCell = 4, demandCell = 5, wastePerSheetCell = 6,
numberPanelPerSheetCell = 7, numberSheetNeededToCut = 8, leftOverPanelCell = 9,
computeTimeCell = 10, xCell = 11, yCell = 12,       overCutPanelCell = 13,
totalWastePerSheetCell = 14, wasteForCuttingCell = 15;
        int r=0;
        int printNo = 0;
        int i=0, j=0, k=0, level = 0;
        int count = 0;
        double totalWastePerSheet = 0;
        //this loop for pick sheet one by one from all sheet
        for (j = 0; j < nPanel.size(); j++) {
          smallestLeftOverArea = sWidth.get(0)*sHeight.get(0);
          row = sheet.createRow(startRow + r);
          for  (i = 0; i < sHeight.size(); i++) {
            numberPanelPerSheet = 0;
            leftoverHeight = sHeight.get(i);
            level = 0;
            System.out.print("\n" + "Sheet: " + (i+1) + " Size: " + sWidth.get(i) + " x " +
sHeight.get(i));
            currentLeftOverArea =  sWidth.get(i) * sHeight.get(i);
            //this loop use to define each level in a specific sheet
            for (double h=leftoverHeight; h>=pHeight.get(pHeight.size()-1);
h=leftoverHeight) {
              leftoverWidth = sWidth.get(i);
              existingLevel = false;
              ArrayList<Double>newPWidth = new ArrayList<Double>();
              ArrayList<Double>newPHeight = new ArrayList<Double>();
              ArrayList<Double>newNPanel = new ArrayList<Double>();
              if (sHeight.get(i) == leftoverHeight) {
                yOrdinate = 0;
              }
              panelHeightAtBeginningLevel.add(pHeight.get(0));
              for (int l = 0; l < pHeight.size(); l++) {
                if (sHeight.get(i)!= leftoverHeight ) {
```

```
            if (pHeight.get(l) <= leftoverHeight && pWidth.get(l) <= leftoverWidth){
              yOrdinate = yOrdinate + panelHeightAtBeginningLevelClone.get(1);

              break;
            }
            if (pHeight.get(l) > leftoverHeight){
              continue;
            }
          }
        }
        level += 1;
        System.out.println();
        System.out.print("\tLevel: " + level + " |");
        //to check the panel in case that it satisfies only one criteria like sample 34

        if (existingLevel == false && ((leftoverHeight>=pHeight.get(j) &&
leftoverWidth>=pWidth.get(j)) )){
          existingLevel = true;
        } else {
          newPWidth = (ArrayList)pWidth.clone();
          newPHeight =(ArrayList)pHeight.clone();
          newNPanel = (ArrayList)nPanel.clone();
          newPWidth.remove(pWidth.size()-1);
          newPHeight.remove(pHeight.size()-1);
          newNPanel.remove(nPanel.size()-1);
        }
        newPWidth.clear();
        newPHeight.clear();
        newNPanel.clear();
        panelHeightAtBeginningLevelClone.clear();
        //Start cutting for the first panel that put into the sheet
        if (leftoverHeight >= pHeight.get(j)) {
          for ( w = leftoverWidth; w >= pWidth.get(j); w = leftoverWidth) {
            if (existingLevel == true && leftoverWidth>=pWidth.get(j)) {
              System.out.print("  " + pWidth.get(j) + " x " + pHeight.get(j) + "  |");
              if (sWidth.get(i) == leftoverWidth) {
                pHeightArray.add(pHeight.get(j));
                pWidthArray.add(pWidth.get(j));
                panelHeightAtBeginningLevel.add(pHeight.get(j));

                xOrdinate = 0;
                xOrdinateArray.add(xOrdinate);
                yOrdinateArray.add(yOrdinate);
              } else {
                pHeightArray.add(pHeight.get(j));
                pWidthArray.add(pWidth.get(j));
                xOrdinate = xOrdinate + pWidthArray.get(pWidthArray.size()-2);
```

```
                    xOrdinateArray.add(xOrdinate);
                    yOrdinateArray.add(yOrdinate);
                  }
                  leftoverWidth = leftoverWidth - pWidth.get(j);
                  currentLeftOverArea = currentLeftOverArea-
(pWidth.get(j)*pHeight.get(j));
                  numberPanelPerSheet += 1;
                  if(usedPanelIndex.indexOf(j) == -1) {
                    usedPanelIndex.add(j);
                    usedPanelAmount.add(1);
                  } else {
                    usedPanelAmount.set(usedPanelIndex.indexOf(j),
usedPanelAmount.get(usedPanelIndex.indexOf(j))+1);
                  }
                }
              }
            }
            //Start cutting for the next panel that put into the sheet when we put other
types of panel in the same level or the next level
            ArrayList<Double> nextPanelCut = new ArrayList<Double>();

            for ( k = j+1; k < pWidth.size(); k++) {
              if(pHeight.get(k) <= leftoverHeight && pWidth.get(k) <= leftoverWidth) {
                nextPanelCut.add((double) k);
                //to pick the new coordinate in the new level
                if (sWidth.get(i) == leftoverWidth) {
                  pHeightArray.add(pHeight.get(k));
                  panelHeightAtBeginningLevel.add(pHeight.get(k));
                  pWidthArray.add(pWidth.get(k));
                  xOrdinate = 0;
                  xOrdinateArray.add(xOrdinate);
                  yOrdinateArray.add(yOrdinate);
                } else {
                  pHeightArray.add(pHeight.get(k));
                  pWidthArray.add(pWidth.get(k));

                  xOrdinate = xOrdinate + pWidthArray.get(pWidthArray.size()-2);
                  xOrdinateArray.add(xOrdinate);
                  yOrdinateArray.add(yOrdinate);
                }
                leftoverWidth -= pWidth.get(k);
                currentLeftOverArea -= (pWidth.get(k)*pHeight.get(k));
                System.out.print("  " + pWidth.get(k) + " x " + pHeight.get(k) + "  |");
                numberPanelPerSheet += 1;
                if(usedPanelIndex.indexOf(k) == -1) {
                  usedPanelIndex.add(k);
                  usedPanelAmount.add(1);
```

```
                } else {
                  usedPanelAmount.set(usedPanelIndex.indexOf(k),
usedPanelAmount.get(usedPanelIndex.indexOf(k))+1);
                }
                k--;
              }
            }
            panelHeightAtBeginningLevelClone =
(ArrayList)panelHeightAtBeginningLevel.clone();
            panelHeightAtBeginningLevel.clear();
            if (leftoverHeight >= pHeight.get(j)) {
              leftoverHeight = leftoverHeight - pHeight.get(j);
            } else {
              if (nextPanelCut.size()!= 0) {
                double a = nextPanelCut.get(0);
                leftoverHeight = leftoverHeight - pHeight.get((int) a);
              }
              if (nextPanelCut.isEmpty()) {
                leftoverHeight = 0;
                continue;
              }
            }
          }
          System.out.println();
          System.out.print("Total Waste for each sheet: " + currentLeftOverArea);
          System.out.print("\nPanel fit: " + numberPanelPerSheet);
          System.out.println();
          if (smallestLeftOverArea > currentLeftOverArea) {
            smallestLeftOverArea = currentLeftOverArea;
            smallestLeftOverSheet = i;
            panelAmountWithMinimumWaste = numberPanelPerSheet;
            pHeightSelected = (ArrayList)pHeightArray.clone();
            pWidthSelected = (ArrayList)pWidthArray.clone();

            xOrdinateSelected = (ArrayList)xOrdinateArray.clone();
            yOrdinateSelected = (ArrayList)yOrdinateArray.clone();

            chosenPanelIndex = (ArrayList)usedPanelIndex.clone();
          }
          for(int ind =0; ind < chosenPanelIndex.size(); ind++) {
            System.out.println("Used Index " + chosenPanelIndex.get(ind));
            System.out.println("Used Amount " + chosenPanelAmount.get(ind));
          }
          // clear it first before selecting the next panel to cut; otherwise, it adds on the
existing data
          pHeightArray.clear();
          pWidthArray.clear();
```

```
            xOrdinateArray.clear();
            yOrdinateArray.clear();

            usedPanelIndex.clear();
            usedPanelAmount.clear();
          }
          System.out.println();
          System.out.println(panelAmountWithMinimumWaste + " panels is cut from
panel set by putting in sheet number " + (smallestLeftOverSheet+1) + " (" +
sWidth.get(smallestLeftOverSheet) + " x " + sHeight.get(smallestLeftOverSheet) + ")
with the minimum waste " + smallestLeftOverArea);

          double minimumSheet = 10000000;
          double result;
          // This loop use to select the pattern that can give the minimum waste to cut.
          for(int d=0; d < chosenPanelIndex.size(); d++) {
            result = Math.ceil(nPanel.get(chosenPanelIndex.get(d)) /
chosenPanelAmount.get(d));
            if (minimumSheet > result) {
              minimumSheet = result;
            }
            System.out.println("To fulfill demand of " +
nPanel.get(chosenPanelIndex.get(d)) + " we need to use " + result + " sheets");
          }
          System.out.println("\nTherefore: To satisfy the demand we need: " +
minimumSheet + " sheets.");
          //To find the over cut panel
          double totalWasteOfOverCutPanel = 0;
          for (int l = 0; l < chosenPanelIndex.size(); l++) {
            overUnderCutPanel.add((minimumSheet * chosenPanelAmount.get(l)) -
nPanel.get(chosenPanelIndex.get(l)));
            if (overUnderCutPanel.get(l)<= 0) {
              overCutPanel.add(0.0);
            } else {
              overCutPanel.add(overUnderCutPanel.get(l));
            }
            wasteForEachOverCutPanelForEachSheet.add(overCutPanel.get(l)*
pHeight.get(l) * pWidth.get(l));
            totalWasteOfOverCutPanel = totalWasteOfOverCutPanel +
wasteForEachOverCutPanelForEachSheet.get(l);
            System.out.println("Over cut Panel = " + overCutPanel);
          }
          totalWastePerSheet = (smallestLeftOverArea * minimumSheet) +
totalWasteOfOverCutPanel;
          totalWastePerSheetArray.add(totalWastePerSheet);
          int index;
```

```
double amount;
double demand;
//Writing the panel of the pattern that we have selected to cut
for(int ind = 0; ind < chosenPanelIndex.size(); ind++) {
  System.out.println("Chosen Index " + chosenPanelIndex.get(ind));
  System.out.println("Used Amount " + chosenPanelAmount.get(ind));
  row = sheet.createRow(startRow + r);

  index = chosenPanelIndex.get(ind);
  amount = chosenPanelAmount.get(ind);
  demand = nPanel.get(chosenPanelIndex.get(ind));
  leftOverDemand.add(demand - (amount * minimumSheet));
  count = 0;
  //Write only that first stage of cutting like No
  if (ind == 0) {
    cell = row.createCell(numberingCell);
    cell.setCellValue(printNo+1);

    cell = row.createCell(sHeightCell);
    cell.setCellValue(sHeight.get(smallestLeftOverSheet));

    cell = row.createCell(sWidthCell);
    cell.setCellValue(sWidth.get(smallestLeftOverSheet));
    if (leftOverDemand.get(ind)> 0) {
      cell = row.createCell(leftOverPanelCell);
      cell.setCellValue(leftOverDemand.get(ind));
    }
  }
  if (ind < chosenPanelIndex.size()-1) {
    cell = row.createCell(demandCell);
    cell.setCellValue(nPanel.get(chosenPanelIndex.get(ind)));

    cell = row.createCell(overCutPanelCell);
    cell.setCellValue(overCutPanel.get(ind));

    cell = row.createCell(numberPanelPerSheetCell);
    cell.setCellValue(chosenPanelAmount.get(ind));
    if (leftOverDemand.get(ind)> 0) {
      cell = row.createCell(leftOverPanelCell);
      cell.setCellValue(leftOverDemand.get(ind));
    }
    // to spread the panel of cutting such that i can put the coordinate
    for (int l = 0; l < chosenPanelAmount.get(ind); l++) {

      cell = row.createCell(pHeightCell);
      cell.setCellValue(pHeightSelected.get(l));
```

```
        cell = row.createCell(pWidthCell);
        cell.setCellValue(pWidthSelected.get(l));

        cell = row.createCell(xCell);
        cell.setCellValue(xOrdinateSelected.get(l));

        cell = row.createCell(yCell);
        cell.setCellValue(yOrdinateSelected.get(l));

        count = count + 1;
        r++;
        row = sheet.createRow(startRow + r);
      }
      //Remove that panel use such that it can print of other panel
      for (int b = (count-1) ; b >= 0; b--) {
        pHeightSelected.remove(b);
        pWidthSelected.remove(b);

        xOrdinateSelected.remove(b);
        yOrdinateSelected.remove(b);
      }
    } else if ((chosenPanelIndex.size()-1) == ind) {
      cell = row.createCell(demandCell);
      cell.setCellValue(nPanel.get(chosenPanelIndex.get(ind)));

      cell = row.createCell(overCutPanelCell);
      cell.setCellValue(overCutPanel.get(ind));

      cell = row.createCell(numberPanelPerSheetCell);
      cell.setCellValue(chosenPanelAmount.get(ind));

      if (leftOverDemand.get(ind)> 0) {
        cell = row.createCell(leftOverPanelCell);
        cell.setCellValue(leftOverDemand.get(ind));
      }
      for (int l = 0; l < chosenPanelAmount.get(ind); l++) {
        cell = row.createCell(pHeightCell);
        cell.setCellValue(pHeightSelected.get(l));

        cell = row.createCell(pWidthCell);
        cell.setCellValue(pWidthSelected.get(l));

        cell = row.createCell(xCell);
        cell.setCellValue(xOrdinateSelected.get(l));

        cell = row.createCell(yCell);
        cell.setCellValue(yOrdinateSelected.get(l));
```

```
            count = count + 1;
            r++;
            row = sheet.createRow(startRow + r);
          }
          for (int b = (count-1) ; b >= 0; b--) {
            pHeightSelected.remove(b);
            pWidthSelected.remove(b);

            xOrdinateSelected.remove(b);
            yOrdinateSelected.remove(b);
          }
          r--;
        }
      }
      wasteForEachOverCutPanelForEachSheet.clear();
      overUnderCutPanel.clear();
      overCutPanel.clear();
      r = r + 2;
      printNo++;
      cell = row.createCell(wastePerSheetCell);
      cell.setCellValue(smallestLeftOverArea);

      cell = row.createCell(numberSheetNeededToCut);
      cell.setCellValue(minimumSheet);

      cell = row.createCell(totalWastePerSheetCell);
      cell.setCellValue(totalWastePerSheet);

      endTime = System.currentTimeMillis();
      System.out.println("Time taken for this process are: " + (endTime - startTime) +
" milli seconds" );

      cell = row.createCell(computeTimeCell);
      cell.setCellValue(endTime - startTime);
      for( int d = chosenPanelIndex.size()-1; d >= 0; d--) {
        index = chosenPanelIndex.indexOf(Collections.max(chosenPanelIndex));
        amount = chosenPanelAmount.get(index);
        demand = nPanel.get(chosenPanelIndex.get(index));
        int newIndex = chosenPanelIndex.get(index);
        if (leftOverDemand.get(index) > 0) {
          nPanel.set(newIndex, leftOverDemand.get(index));
          chosenPanelIndex.remove(index);
          chosenPanelAmount.remove(index);
          leftOverDemand.remove(index);
        } else {
          nPanel.remove(newIndex);
```

```
            pHeight.remove(newIndex);
            pWidth.remove(newIndex);
            chosenPanelIndex.remove(index);
            chosenPanelAmount.remove(index);
            leftOverDemand.remove(index);
          }
        }
        leftOverDemand.clear();
        System.out.println("----------------------------------------------------------------");
        j--;
        double wasteForCutting = 0;
        for (int l = 0; l < totalWastePerSheetArray.size(); l++) {
          wasteForCutting = wasteForCutting + totalWastePerSheetArray.get(l);
        }
        cell = row.createCell(wasteForCuttingCell);
        cell.setCellValue(wasteForCutting);
      }
      chosenPanelIndex.clear();
      chosenPanelAmount.clear();
      file.close();

      FileOutputStream outFile =new FileOutputStream(new File(excelFile));
      workbook.write(outFile);
      outFile.close();
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
  public static void ReadExcel(ArrayList<Double> sWidth,
    ArrayList<Double> sHeight, ArrayList<Double> nSheet,
    ArrayList<Double> pWidth, ArrayList<Double> pHeight,
    ArrayList<Double> nPanel, String excelFile) {
    try {
      FileInputStream file = new FileInputStream(new File(excelFile));
      XSSFWorkbook workbook = new XSSFWorkbook(file);
      XSSFSheet sheet = workbook.getSheet("Input");// or .getsheetAt(2);
      // cRow = currentRow, pRow = processingRow
      int cRow = 0 , pRow = 0;
      int cColumn = 1;
      int dataInRow = 0;
      //Iterate through each rows from first sheet
      Iterator<Row> rowIterator = sheet.iterator();
      while(rowIterator.hasNext()) {
        Row row = rowIterator.next();
        cRow++;
```

```
cColumn = 1;
dataInRow = 0;

//For each row, iterate through each columns
Iterator<Cell> cellIterator = row.cellIterator();
while(cellIterator.hasNext()) {
  Cell cell = cellIterator.next();
  switch(cell.getCellType()) {
  case Cell.CELL_TYPE_BOOLEAN:
    System.out.print(cell.getBooleanCellValue() + "\t\t");
    dataInRow ++;
    break;
  case Cell.CELL_TYPE_NUMERIC:
    System.out.print(cell.getNumericCellValue() + "\t\t");
    if(cRow>pRow) {
      pRow = cRow;
    }
    else
    {
      switch(cColumn) {
      case 1:
        pHeight.add(cell.getNumericCellValue());
        break;
      case 2:
        pWidth.add(cell.getNumericCellValue());
        break;
      case 3:
        nPanel.add(cell.getNumericCellValue());
        break;
      case 4:
        sHeight.add(cell.getNumericCellValue());
        break;
      case 5:
        sWidth.add(cell.getNumericCellValue());
        break;
      case 6:
        nSheet.add(cell.getNumericCellValue());
      }
      cColumn++;
    }
    dataInRow++;
    break;
  case Cell.CELL_TYPE_STRING:
    System.out.print(cell.getStringCellValue() + "\t\t");
    dataInRow++;
    break;
  }
```

```
        }
        if(dataInRow > 0) {
          System.out.println("");
        }
      }
      file.close();
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

7.1.5. 2D Vertical Improvement

```
package Simple_Heuristic_II;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class Improve2_Simple_Heuristic_P_WidthBased_Cut_Vertical{
  public static void main(String[] args) {
    long startTime = System.currentTimeMillis();
    ArrayList<Double> sWidth = new ArrayList<Double>();
    ArrayList<Double> sHeight = new ArrayList<Double>();
    ArrayList<Double> nSheet = new ArrayList<Double>();

    ArrayList<Double> pWidth = new ArrayList<Double>();
    ArrayList<Double> pHeight = new ArrayList<Double>();
    ArrayList<Double> nPanel = new ArrayList<Double>();
    ArrayList<Double> sLeftOverArea = new ArrayList<Double>();
    String excelFile = "D:\\Input and output of heuristics based on
height_ExactNumber.xlsx";
    System.out.println("ArrayList in row and column set: ");
    ReadExcel(sWidth, sHeight, nSheet, pWidth, pHeight, nPanel, excelFile);

    ArrayList<Double> sheetWidth = new ArrayList<Double>();
    ArrayList<Double> sheetHeight = new ArrayList<Double>();

    ArrayList<Double> panelWidth = new ArrayList<Double>();
    ArrayList<Double> panelHeight = new ArrayList<Double>();

    ArrayList<Double> overCutPanel = new ArrayList<Double>();
    ArrayList<Double> overUnderCutPanel = new ArrayList<Double>();

    ArrayList<Double> wasteForEachOverCutPanelForEachSheet = new
ArrayList<Double>();
    ArrayList<Double> totalWastePerSheetArray = new ArrayList<Double>();
```

```
    //define sheet set
    System.out.println("\n" + "Sheet set: ");
    for (int i=0; i<nSheet.size(); i++) {
      for (int a=0; a<nSheet.get(i); a++) {
        sheetWidth.add(sWidth.get(i));
        sheetHeight.add(sHeight.get(i));
        sLeftOverArea.add(sWidth.get(i) * sHeight.get(i));
        System.out.print( "[" + sHeight.get(i) + " x " + sWidth.get(i) + "]" + ", ");
      }System.out.println();
    }
    //define panel set
    System.out.println("\n" + "Panel set:");
    for (int i=0; i<nPanel.size(); i++) {
      for (int a=0; a<nPanel.get(i); a++) {
        panelWidth.add(pWidth.get(i));
        panelHeight.add(pHeight.get(i));
        System.out.print( "[" + pHeight.get(i) + " x " + pWidth.get(i)+ "]" + ", ");
      }System.out.println();
    }
    // To check and remove the big panel that we cannot put into the sheet, if we don't
have it, it might cause to error.
    double maxSHeight = 0;
    double maxSWidth = 0;
    for (int j = 0; j < nPanel.size(); j++) {
      for (int i = 0; i < nSheet.size(); i++) {
        if (sHeight.get(i) > maxSHeight) {
          maxSHeight = sHeight.get(i);
        }
        if (sWidth.get(i) > maxSWidth) {
          maxSWidth = sWidth.get(i);
        }
      }
      if (((pHeight.get(j) > maxSHeight)) || (pWidth.get(j) > maxSWidth)) {
        pWidth.remove(j);
        pHeight.remove(j);
        nPanel.remove(j);
        j--;
      }
    }
    //To check and remove the small sheet that we cannot use to cut other panel, if we
don't have it, it might cause to error
    double minPHeight = 999999999;
    double minPWidth = 999999999;
    for (int i = 0; i < nSheet.size(); i++) {
      for (int j = 0; j < nPanel.size(); j++) {
        if (minPHeight > pHeight.get(j)) {
          minPHeight = pHeight.get(j);
```

```
    }
    if (minPWidth > pWidth.get(j)) {
      minPWidth = pWidth.get(j);
    }
  }
  if ((sHeight.get(i) < minPHeight) || (sWidth.get(i) < minPWidth)) {
    sWidth.remove(i);
    sHeight.remove(i);
    nSheet.remove(i);
    i--;
  }
}
//here is the process of cutting
try {
  FileInputStream file = new FileInputStream(new File(excelFile));
  XSSFWorkbook workbook = new XSSFWorkbook(file);
  XSSFSheet sheet = workbook.getSheet("Output2");
  Row row; // Declare row variable as Excel row.
  Cell cell;  // Declare cell variable as Excel cell.

  System.out.println("\n" + "Here is the step of cutting: ");
  ArrayList<Integer> chosenPanelIndex = new ArrayList<Integer>();
  ArrayList<Integer> chosenPanelAmount = new ArrayList<Integer>();

  ArrayList<Integer> usedPanelIndex = new ArrayList<Integer>();
  ArrayList<Integer> usedPanelAmount = new ArrayList<Integer>();
  ArrayList<Double> leftOverDemand = new ArrayList<Double>();

  ArrayList<Double> xOrdinateArray = new ArrayList<Double>();
  ArrayList<Double> yOrdinateArray = new ArrayList<Double>();

  ArrayList<Double> xOrdinateSelected = new ArrayList<Double>();
  ArrayList<Double> yOrdinateSelected = new ArrayList<Double>();

  ArrayList<Double> pHeightArray = new ArrayList<Double>();
  ArrayList<Double> pWidthArray = new ArrayList<Double>();

  ArrayList<Double> pHeightSelected = new ArrayList<Double>();
  ArrayList<Double> pWidthSelected = new ArrayList<Double>();

  ArrayList<Double> panelHeightAtBeginningLevel = new ArrayList<Double>();
  ArrayList<Double> panelHeightAtBeginningLevelClone = new
  ArrayList<Double>();

  double leftoverHeight;
  double leftoverWidth;
  boolean existingLevel = false;
```

```
        double currentLeftOverArea=0;
        double smallestLeftOverArea=0;
        int smallestLeftOverSheet=0;
        int numberPanelPerSheet=0;
        int panelAmountWithMinimumWaste=0;
        double sheetNumberNeeded;
        long endTime = 0;
        double xOrdinate = 0;
        double yOrdinate = 0;
        double w = 0;
        int startRow = 2, numberingCell = 0, sHeightCell = 1, sWidthCell = 2,
pHeightCell = 3, pWidthCell = 4, demandCell = 5, wastePerSheetCell = 6,
numberPanelPerSheetCell = 7, numberSheetNeededToCut = 8, leftOverPanelCell = 9,
computeTimeCell = 10, xCell = 11, yCell = 12,       overCutPanelCell = 13,
totalWastePerSheetCell = 14, wasteForCuttingCell = 15;
        int r=0;
        int printNo = 0;
        int i=0, j=0, k=0, level = 0;
        int count = 0;
        double totalWastePerSheet = 0;
        //this loop for pick sheet one by one from all sheet
        for (j = 0; j < nPanel.size(); j++) {
          smallestLeftOverArea = sWidth.get(0)*sHeight.get(0);
          row = sheet.createRow(startRow + r);
          for  (i = 0; i < sHeight.size(); i++) {
            numberPanelPerSheet = 0;
            leftoverWidth = sWidth.get(i);
            level = 0;
            System.out.print("\n" + "Sheet: " + (i+1) + " Size: " + sWidth.get(i) + " x " +
sHeight.get(i));
            currentLeftOverArea =  sWidth.get(i) * sHeight.get(i);

            //this loop use to define each level in a specific sheet
            for (double h=leftoverWidth; h>=pWidth.get(pWidth.size()-1);
h=leftoverWidth) {
              leftoverHeight = sHeight.get(i);
              existingLevel = false;
              ArrayList<Double>newPWidth = new ArrayList<Double>();
              ArrayList<Double>newPHeight = new ArrayList<Double>();
              ArrayList<Double>newNPanel = new ArrayList<Double>();

              if (sWidth.get(i) == leftoverWidth) {
                xOrdinate = 0;
              }
            panelHeightAtBeginningLevel.add(pWidth.get(0));
            for (int l = 0; l < pWidth.size(); l++) {
              if (sWidth.get(i)!= leftoverWidth) {
```

```
            if (pWidth.get(l) <= leftoverWidth && pHeight.get(l) <= leftoverHeight) {
              xOrdinate = xOrdinate + panelHeightAtBeginningLevelClone.get(1);

              break;
            }
            if (pWidth.get(l) > leftoverWidth){
              continue;
            }
          }
        }
        level += 1;
        System.out.println();
        System.out.print("\tLevel: " + level + " |");
        if (existingLevel == false && ((leftoverHeight>=pHeight.get(j) &&
leftoverWidth>=pWidth.get(j)) )){
          existingLevel = true;
        } else {
          newPWidth = (ArrayList)pWidth.clone();
          newPHeight =(ArrayList)pHeight.clone();
          newNPanel = (ArrayList)nPanel.clone();

          newPWidth.remove(pWidth.size()-1);
          newPHeight.remove(pHeight.size()-1);
          newNPanel.remove(nPanel.size()-1);
        }
        newPWidth.clear();
        newPHeight.clear();
        newNPanel.clear();
        panelHeightAtBeginningLevelClone.clear();
        if (leftoverWidth >= pWidth.get(j)) {
          for (w = leftoverHeight; w >= pHeight.get(j); w = leftoverHeight) {
            if (existingLevel == true && leftoverHeight >= pHeight.get(j)) {
              System.out.print("  " + pWidth.get(j) + " x " + pHeight.get(j) + "  |");
              if (sHeight.get(i) == leftoverHeight) {
                pHeightArray.add(pHeight.get(j));
                pWidthArray.add(pWidth.get(j));
                panelHeightAtBeginningLevel.add(pWidth.get(j));

                yOrdinate = 0;
                xOrdinateArray.add(xOrdinate);
                yOrdinateArray.add(yOrdinate);
              } else {
                pHeightArray.add(pHeight.get(j));
                pWidthArray.add(pWidth.get(j));
                yOrdinate = yOrdinate + pHeightArray.get(pHeightArray.size()-2);
                xOrdinateArray.add(xOrdinate);
                yOrdinateArray.add(yOrdinate);
```

```
                }
                leftoverHeight = leftoverHeight - pHeight.get(j);
                currentLeftOverArea = currentLeftOverArea-
        (pWidth.get(j)*pHeight.get(j));
                numberPanelPerSheet += 1;
                if(usedPanelIndex.indexOf(j) == -1) {
                  usedPanelIndex.add(j);
                  usedPanelAmount.add(1);
                } else {
                  usedPanelAmount.set(usedPanelIndex.indexOf(j),
    usedPanelAmount.get(usedPanelIndex.indexOf(j))+1);
                  //use to increase the amount of the same panel size that put into the sheet
                }
              }
            }
          }
          ArrayList<Double> nextPanelCut = new ArrayList<Double>();
          for ( k = j+1; k < pHeight.size(); k++) {
            if(pHeight.get(k) <= leftoverHeight && pWidth.get(k) <= leftoverWidth) {
              nextPanelCut.add((double) k);
              if (sHeight.get(i) == leftoverHeight) {

                pHeightArray.add(pHeight.get(k));

                panelHeightAtBeginningLevel.add(pWidth.get(k));
                pWidthArray.add(pWidth.get(k));

                yOrdinate = 0;
                xOrdinateArray.add(xOrdinate);
                yOrdinateArray.add(yOrdinate);
              } else {
                pHeightArray.add(pHeight.get(k));
                pWidthArray.add(pWidth.get(k));
                yOrdinate = yOrdinate + pHeightArray.get(pHeightArray.size()-2);

                xOrdinateArray.add(xOrdinate);
                yOrdinateArray.add(yOrdinate);
              }
              leftoverHeight -= pHeight.get(k)
              currentLeftOverArea -= (pWidth.get(k) * pHeight.get(k));

              System.out.print("  " + pWidth.get(k) + " x " + pHeight.get(k) + "  |");
              numberPanelPerSheet += 1;
              if(usedPanelIndex.indexOf(k) == -1) {
                usedPanelIndex.add(k);
                usedPanelAmount.add(1);
              } else {
```

```
                usedPanelAmount.set(usedPanelIndex.indexOf(k),
        usedPanelAmount.get(usedPanelIndex.indexOf(k))+1);
              }
              k--;
            }
          }
          panelHeightAtBeginningLevelClone =
(ArrayList)panelHeightAtBeginningLevel.clone();
          panelHeightAtBeginningLevel.clear();
          if (leftoverWidth >= pWidth.get(j)) {
            leftoverWidth = leftoverWidth - pWidth.get(j);
          } else {
            if (nextPanelCut.size()!= 0) {
              double a = nextPanelCut.get(0);
              leftoverWidth = leftoverWidth - pWidth.get((int) a);
            }
            if (nextPanelCut.isEmpty()) {
              leftoverWidth = 0;
              continue;
            }
          }
        }
        System.out.println();
        System.out.print("Total Waste for each sheet: " + currentLeftOverArea);
        System.out.print("\nPanel fit: " + numberPanelPerSheet);
        System.out.println();
        if (smallestLeftOverArea > currentLeftOverArea) {
          smallestLeftOverArea = currentLeftOverArea;
          smallestLeftOverSheet = i;
          panelAmountWithMinimumWaste = numberPanelPerSheet;

          pHeightSelected = (ArrayList)pHeightArray.clone();
          pWidthSelected = (ArrayList)pWidthArray.clone();

          xOrdinateSelected = (ArrayList)xOrdinateArray.clone();
          yOrdinateSelected = (ArrayList)yOrdinateArray.clone();

          chosenPanelIndex = (ArrayList)usedPanelIndex.clone();
          chosenPanelAmount = (ArrayList)usedPanelAmount.clone();
        }
        for (int ind =0; ind < chosenPanelIndex.size(); ind++) {
          System.out.println("Used Index " + chosenPanelIndex.get(ind));
          System.out.println("Used Amount " + chosenPanelAmount.get(ind));
        }
        pHeightArray.clear();
        pWidthArray.clear();
```

```
        xOrdinateArray.clear();
        yOrdinateArray.clear();

        usedPanelIndex.clear();
        usedPanelAmount.clear();
      }
      System.out.println();
      System.out.println(panelAmountWithMinimumWaste + " panels is cut from
panel set by putting in sheet number " + (smallestLeftOverSheet+1) + " (" +
sWidth.get(smallestLeftOverSheet) + " x " + sHeight.get(smallestLeftOverSheet) + ")
with the minimum waste " + smallestLeftOverArea);
      double minimumSheet = 10000000;
      double result;
      for(int d=0; d < chosenPanelIndex.size(); d++) {
        result = Math.ceil(nPanel.get(chosenPanelIndex.get(d)) /
chosenPanelAmount.get(d));
        if (minimumSheet > result) {
          minimumSheet = result;
        }
        System.out.println("To fulfill demand of " +
nPanel.get(chosenPanelIndex.get(d)) + " we need to use " + result + " sheets");
      }
      System.out.println("\nTherefore: To satisfy the demand we need: " +
minimumSheet + " sheets.");
      //To find the over cut panel
      double totalWasteOfOverCutPanel = 0;
      for (int l = 0; l < chosenPanelIndex.size(); l++) {
        overUnderCutPanel.add((minimumSheet * chosenPanelAmount.get(l)) -
nPanel.get(chosenPanelIndex.get(l)));
        if (overUnderCutPanel.get(l)<= 0) {
          overCutPanel.add(0.0);
        } else {
          overCutPanel.add(overUnderCutPanel.get(l));
        }
        wasteForEachOverCutPanelForEachSheet.add(overCutPanel.get(l)*
pHeight.get(l) * pWidth.get(l));
        totalWasteOfOverCutPanel = totalWasteOfOverCutPanel +
wasteForEachOverCutPanelForEachSheet.get(l);
        System.out.println("Over cut Panel = " + overCutPanel);
      }
      totalWastePerSheet = (smallestLeftOverArea * minimumSheet) +
totalWasteOfOverCutPanel;
      totalWastePerSheetArray.add(totalWastePerSheet);
      int index;
      double amount;
      double demand;
      for(int ind = 0; ind < chosenPanelIndex.size(); ind++) {
```

```
System.out.println("Chosen Index " + chosenPanelIndex.get(ind));
System.out.println("Used Amount " + chosenPanelAmount.get(ind));
row = sheet.createRow(startRow + r);

index = chosenPanelIndex.get(ind);
amount = chosenPanelAmount.get(ind);
demand = nPanel.get(chosenPanelIndex.get(ind));
leftOverDemand.add(demand - (amount * minimumSheet));
count = 0;
if (ind == 0) {
  cell = row.createCell(numberingCell);
  cell.setCellValue(printNo+1);

  cell = row.createCell(sHeightCell);
  cell.setCellValue(sHeight.get(smallestLeftOverSheet));

  cell = row.createCell(sWidthCell);
  cell.setCellValue(sWidth.get(smallestLeftOverSheet));

  if (leftOverDemand.get(ind)> 0) {
    cell = row.createCell(leftOverPanelCell);
    cell.setCellValue(leftOverDemand.get(ind));
  }
}
if (ind < chosenPanelIndex.size()-1) {
  cell = row.createCell(demandCell);
  cell.setCellValue(nPanel.get(chosenPanelIndex.get(ind)));

  cell = row.createCell(overCutPanelCell);
  cell.setCellValue(overCutPanel.get(ind));

  cell = row.createCell(numberPanelPerSheetCell);
  cell.setCellValue(chosenPanelAmount.get(ind));
  if (leftOverDemand.get(ind)> 0) {
    cell = row.createCell(leftOverPanelCell);
    cell.setCellValue(leftOverDemand.get(ind));
  }
  // to spread the panel of cutting such that i can put the coordinate
  for (int l = 0; l < chosenPanelAmount.get(ind); l++) {

    cell = row.createCell(pHeightCell);
    cell.setCellValue(pHeightSelected.get(l));

    cell = row.createCell(pWidthCell);
    cell.setCellValue(pWidthSelected.get(l));

    cell = row.createCell(xCell);
```

```
    cell.setCellValue(xOrdinateSelected.get(l));

    cell = row.createCell(yCell);
    cell.setCellValue(yOrdinateSelected.get(l));

    count = count + 1;
    r++;
    row = sheet.createRow(startRow + r);
   }
   //Remove that panel use such that it can print of other panel
   for (int b = (count-1) ; b >= 0; b--) {
    pHeightSelected.remove(b);
    pWidthSelected.remove(b);

    xOrdinateSelected.remove(b);
    yOrdinateSelected.remove(b);
   }
  } else if ((chosenPanelIndex.size()-1) == ind) {
   cell = row.createCell(demandCell);
   cell.setCellValue(nPanel.get(chosenPanelIndex.get(ind)));

   cell = row.createCell(overCutPanelCell);
   cell.setCellValue(overCutPanel.get(ind));

   cell = row.createCell(numberPanelPerSheetCell);
   cell.setCellValue(chosenPanelAmount.get(ind));

   if (leftOverDemand.get(ind)> 0) {
    cell = row.createCell(leftOverPanelCell);
    cell.setCellValue(leftOverDemand.get(ind));
   }
   for (int l = 0; l < chosenPanelAmount.get(ind); l++) {
    cell = row.createCell(pHeightCell);
    cell.setCellValue(pHeightSelected.get(l));

    cell = row.createCell(pWidthCell);
    cell.setCellValue(pWidthSelected.get(l));

    cell = row.createCell(xCell);
    cell.setCellValue(xOrdinateSelected.get(l));

    cell = row.createCell(yCell);
    cell.setCellValue(yOrdinateSelected.get(l));

    count = count + 1;
    r++;
    row = sheet.createRow(startRow + r);
```

```
            }
            for (int b = (count-1) ; b >= 0; b--) {
              pHeightSelected.remove(b);
              pWidthSelected.remove(b);

              xOrdinateSelected.remove(b);
              yOrdinateSelected.remove(b);
            }
            r--;
          }
        }
        wasteForEachOverCutPanelForEachSheet.clear();
        overUnderCutPanel.clear();
        overCutPanel.clear();
        r = r + 2;
        printNo++;

        cell = row.createCell(wastePerSheetCell);
        cell.setCellValue(smallestLeftOverArea);

        cell = row.createCell(numberSheetNeededToCut);
        cell.setCellValue(minimumSheet);

        cell = row.createCell(totalWastePerSheetCell);
        cell.setCellValue(totalWastePerSheet);

        endTime = System.currentTimeMillis();
        System.out.println("Time taken for this process are: " + (endTime - startTime) +
" milli seconds" );

        cell = row.createCell(computeTimeCell);
        cell.setCellValue(endTime - startTime);

        for( int d = chosenPanelIndex.size()-1; d >= 0; d--) {
          index = chosenPanelIndex.indexOf(Collections.max(chosenPanelIndex));
          amount = chosenPanelAmount.get(index);
          demand = nPanel.get(chosenPanelIndex.get(index));
          int newIndex = chosenPanelIndex.get(index);
          if (leftOverDemand.get(index) > 0) {
            nPanel.set(newIndex, leftOverDemand.get(index));
            chosenPanelIndex.remove(index);
            chosenPanelAmount.remove(index);
            leftOverDemand.remove(index);
          } else {
            nPanel.remove(newIndex);
            pHeight.remove(newIndex);
            pWidth.remove(newIndex);
```

```
            chosenPanelIndex.remove(index);
            chosenPanelAmount.remove(index);
            leftOverDemand.remove(index);
          }
        }
        leftOverDemand.clear();
        System.out.println("----------------------------------------------------------------------");
        j--;
        double wasteForCutting = 0;
        for (int l = 0; l < totalWastePerSheetArray.size(); l++) {
          wasteForCutting = wasteForCutting + totalWastePerSheetArray.get(l);
        }
        cell = row.createCell(wasteForCuttingCell);
        cell.setCellValue(wasteForCutting);
      }
      chosenPanelIndex.clear();
      chosenPanelAmount.clear();
      file.close();

      FileOutputStream outFile =new FileOutputStream(new File(excelFile));
      workbook.write(outFile);
      outFile.close();
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
      }
    }
    public static void ReadExcel(ArrayList<Double> sWidth,
      ArrayList<Double> sHeight, ArrayList<Double> nSheet,
      ArrayList<Double> pWidth, ArrayList<Double> pHeight,
      ArrayList<Double> nPanel, String excelFile) {
      try {
        FileInputStream file = new FileInputStream(new File(excelFile));
        XSSFWorkbook workbook = new XSSFWorkbook(file);
        XSSFSheet sheet = workbook.getSheet("Input2");
        // cRow = currentRow, pRow = processingRow
        int cRow = 0 , pRow = 0;
        int cColumn = 1;
        int dataInRow = 0;

        //Iterate through each rows from first sheet
        Iterator<Row> rowIterator = sheet.iterator();
        while(rowIterator.hasNext()) {
          Row row = rowIterator.next();
          cRow++;
          cColumn = 1;
```

```
dataInRow = 0;
//For each row, iterate through each columns
Iterator<Cell> cellIterator = row.cellIterator();
while(cellIterator.hasNext()) {
  Cell cell = cellIterator.next();
  switch(cell.getCellType()) {
  case Cell.CELL_TYPE_BOOLEAN:
    System.out.print(cell.getBooleanCellValue() + "\t\t");
    dataInRow ++;
    break;
  case Cell.CELL_TYPE_NUMERIC:
    System.out.print(cell.getNumericCellValue() + "\t\t");
    if(cRow>pRow) {
      pRow = cRow;
    }
    else
    {
      switch(cColumn) {
    case 1:
      pHeight.add(cell.getNumericCellValue());
      break;
    case 2:
      pWidth.add(cell.getNumericCellValue());
      break;
    case 3:
      nPanel.add(cell.getNumericCellValue());
      break;
    case 4:
      sHeight.add(cell.getNumericCellValue());
      break;
    case 5:
      sWidth.add(cell.getNumericCellValue());
      break;
    case 6:
      nSheet.add(cell.getNumericCellValue());
    }
    cColumn++;
  }
  dataInRow++;
  break;
  case Cell.CELL_TYPE_STRING:
    System.out.print(cell.getStringCellValue() + "\t\t");
    dataInRow++;
    break;
  }
}
if(dataInRow > 0) {
```

```
            System.out.println("");
          }
        }
        file.close();
      } catch (FileNotFoundException e) {
        e.printStackTrace();
      } catch (IOException e) {
        e.printStackTrace();
      }
    }
}
```

7.1.6. Sheet Width Panel Height Horizontal Cut

```
package SWidthBased_PHeightBased_2D_2Stage_Horizontal_Cutting_P3;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class SWidthBased_PHeightBased_2D_2Stage_Horizontal_Cutting_P3{
  public static void main(String[] args) {
    long startTime = System.currentTimeMillis();
    ArrayList<Double> sWidth = new ArrayList<Double>();
    ArrayList<Double> sHeight = new ArrayList<Double>();   // Unique sheet height.
    ArrayList<Double> nSheet = new ArrayList<Double>();   // Amount of sheet of
each size.

    ArrayList<Double> pWidth = new ArrayList<Double>();
    ArrayList<Double> pHeight = new ArrayList<Double>();
    ArrayList<Double> nPanel = new ArrayList<Double>();

    ArrayList<Double> sLeftOverArea = new ArrayList<Double>();
    String excelFile = "C:\\Users\\Tiengkimseng\\Google
Drive\\Kimseng\\Thesis\\CODE_2D_2S_AND_2D_3S_CUTTING_8MODELS\\Heu
ristic_for_2D_2Stage_Con1_Con2_P1_P3\\SWidth_PHeight_P3\\CuttingData -
Sample 39.xlsx";
    //String excelFile = "D:\\Input and output of heuristics For construction 1 and
2.xlsx";
    System.out.println("ArrayList in row and column set: ");
    try {
      FileInputStream file = new FileInputStream(new File(excelFile));

      XSSFWorkbook workbook = new XSSFWorkbook(file);
      XSSFSheet sheet = workbook.getSheet("Input");// or .getsheetAt(2);
      // cRow = currentRow, pRow = processingRow
      int cRow = 0 , pRow = 0;          // cRow can give any value but pRow cannot
      int cColumn = 1;            // here we also can change it
      int dataInRow = 0;
```

```
//Iterate through each rows from first sheet
Iterator<Row> rowIterator = sheet.iterator();
while(rowIterator.hasNext()) {
  Row row = rowIterator.next();
  cRow++;
  cColumn = 1;
  dataInRow = 0;

  //For each row, iterate through each columns
  Iterator<Cell> cellIterator = row.cellIterator();
  while(cellIterator.hasNext()) {
    Cell cell = cellIterator.next();
    switch(cell.getCellType()) {
    case Cell.CELL_TYPE_BOOLEAN:
      System.out.print(cell.getBooleanCellValue() + "\t\t");
      dataInRow ++;
      break;
    case Cell.CELL_TYPE_NUMERIC:
      System.out.print(cell.getNumericCellValue() + "\t\t");
      if(cRow>pRow) {
        pRow = cRow;
      }
      Else
      {
        switch(cColumn) {
        case 1:
          pHeight.add(cell.getNumericCellValue());
          break;
        case 2:
          pWidth.add(cell.getNumericCellValue());
          break;
        case 3:
          nPanel.add(cell.getNumericCellValue());
          break;
        case 4:
          sHeight.add(cell.getNumericCellValue());
          break;
        case 5:
          sWidth.add(cell.getNumericCellValue());
          break;
        case 6:
          nSheet.add(cell.getNumericCellValue());
        }
        cColumn++;
      }
      dataInRow++;
      break;
```

```
      case Cell.CELL_TYPE_STRING:
        System.out.print(cell.getStringCellValue() + "\t\t");
        dataInRow++;
        break;
      }
    }
    if(dataInRow > 0) {
      System.out.println("");
    }
  }
  file.close();
} catch (FileNotFoundException e) {
  e.printStackTrace();
} catch (IOException e) {
  e.printStackTrace();
}
ArrayList<Double> sheetWidth = new ArrayList<Double>();
ArrayList<Double> sheetHeight = new ArrayList<Double>();

ArrayList<Double> panelWidth = new ArrayList<Double>();
ArrayList<Double> panelHeight = new ArrayList<Double>();

ArrayList<Double> totalWastePerSheet = new ArrayList<Double>();
//define sheet set
System.out.println("\n" + "Sheet set: ");
for (int i=0; i<nSheet.size(); i++) {
  for (int a=0; a<nSheet.get(i); a++) {
    sheetWidth.add(sWidth.get(i));
    sheetHeight.add(sHeight.get(i));
    sLeftOverArea.add(sWidth.get(i) * sHeight.get(i));
    //System.out.print( "[" + sHeight.get(i) + " x " + sWidth.get(i) + "]" + ", ");
  }System.out.println();
}
//define panel set
System.out.println("\n" + "Panel set:");
for (int i=0; i<nPanel.size(); i++) {
  for (int a=0; a<nPanel.get(i); a++) {
    panelWidth.add(pWidth.get(i));
    panelHeight.add(pHeight.get(i));
    //System.out.print( "[" + pHeight.get(i) + " x " + pWidth.get(i)+ "]" + ", ");
  }System.out.println();
}
//here is the process of cutting
try {
  // Open excel file.
  FileInputStream file = new FileInputStream(new File(excelFile));
```

```
        XSSFWorkbook workbook = new XSSFWorkbook(file);
        XSSFSheet sheet = workbook.getSheet("Output");// or .getsheetAt(2);

        Row row; // Declare row variable as Excel row.
        Cell cell;  // Declare cell variable as Excel cell.

        System.out.println("\n" + "Here is the step of cutting: ");
        double leftoverHeight;
        double leftoverWidth;
        double wasteBeforeCut = 0;
        // Now everything is still working as expected. So no other code editing is
necessary.
        int startRow = 2, sHeightCell = 0, sWidthCell = 1, pHeightCell = 2, pWidthCell
= 3, levelCell = 4, sLeftOverAreaCell = 5, computeTimeCell = 6, totalWasteCell = 7;
        int r=0;
        long computeTime = 0;
        boolean existingLevel = false;  // Level check if new calculate new
[leftoverHeight] and [leftoverWidth].
        int i=0, j=0, level;
        //.size equal to .length but one use in arraylist and another one use in array
        this loop for pick sheet one by one from all sheet
        for (i = 0; i < sheetHeight.size(); i++) {
          leftoverHeight = sheetHeight.get(i);
          leftoverWidth = sheetWidth.get(i);//.get(i) use in arraylist
          level = 0;

          row = sheet.createRow(startRow + r);

          cell = row.createCell(sHeightCell);
          cell.setCellValue(sheetHeight.get(i));

          cell = row.createCell(sWidthCell);
          cell.setCellValue(sheetWidth.get(i));
          //To define the value of the waste
          if (i > 0) {
            sheet.getRow(startRow + (r-
1)).createCell(sLeftOverAreaCell).setCellValue(sLeftOverArea.get(i-1));
          }
          //this loop use to define each level in a specific sheet
          for (double h=leftoverHeight; h>=panelHeight.get(panelHeight.size()-1);
h=leftoverHeight) {
            leftoverWidth = sheetWidth.get(i);
            existingLevel = false;
            // this loop use to pick the panel to put inside each level in a specific sheet
            for (j = 0; j < panelHeight.size(); j++) {
              if (existingLevel == false && (leftoverHeight>=panelHeight.get(j) &&
leftoverWidth>=panelWidth.get(j))) {
```

```
            leftoverHeight = leftoverHeight - panelHeight.get(j);
            existingLevel = true;
            level += 1;
          }
          if (existingLevel == true && leftoverWidth >= panelWidth.get(j)) {
            System.out.println("put panel: " + panelHeight.get(j) + " x " +
panelWidth.get(j) + " into sheet: " + (i+1) + " level: " + level);
            sLeftOverArea.set(i, sLeftOverArea.get(i) - (panelHeight.get(j) *
panelWidth.get(j)));
            try {
              row = sheet.getRow(startRow + r);
              if (row == null) {
                row = sheet.createRow(startRow + r);
              }
            }
            catch (Exception e) {
              row = sheet.createRow(startRow + r);
            }
            row.createCell(pHeightCell).setCellValue(panelHeight.get(j));
            row.createCell(pWidthCell).setCellValue(panelWidth.get(j));
            row.createCell(levelCell).setCellValue(level);

            r = r + 1;     // move to new row.
            leftoverWidth = leftoverWidth - panelWidth.get(j);
            panelHeight.remove(j);
            panelWidth.remove(j);
            j = j - 1;
          }
        }
        if(panelHeight.isEmpty()){
          sheet.getRow(startRow + (r-
1)).createCell(sLeftOverAreaCell).setCellValue(sLeftOverArea.get(i));
          sheet.getRow(startRow + (r-
1)).createCell(computeTimeCell).setCellValue(computeTime);
          totalWastePerSheet.add(sLeftOverArea.get(i));
          for (int k = 0; k < totalWastePerSheet.size(); k++) {
            wasteBeforeCut = wasteBeforeCut + totalWastePerSheet.get(k);
          }
          sheet.getRow(startRow + (r-
1)).createCell(totalWasteCell).setCellValue(wasteBeforeCut);
          file.close();
          FileOutputStream outFile =new FileOutputStream(new File(excelFile));
          workbook.write(outFile);
          outFile.close();
          return;
        }
      }
```

```
        totalWastePerSheet.add(sLeftOverArea.get(i));
        sheetHeight.remove(0);
        sheetWidth.remove(0);
        long endTime = System.currentTimeMillis();
        computeTime = endTime - startTime;
        System.out.println("Time taken for this process are: " + (computeTime) + "
milli seconds" );
        cell = row.createCell(computeTimeCell);
        cell.setCellValue(computeTime);
      }
      sheet.getRow(startRow + (r-
1)).createCell(sLeftOverAreaCell).setCellValue(sLeftOverArea.get(9));
      file.close();
      FileOutputStream outFile =new FileOutputStream(new File(excelFile));
      workbook.write(outFile);
      outFile.close();
      for(int l=0; l < sLeftOverArea.size(); l++) {
        System.out.println("Sheet: " + (l+1) + " Left over area: " +
sLeftOverArea.get(l));
      }
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

7.1.7. Sheet Width Panel Width Vertical Cut

```
package SWidthBased_PWidthBased_2D_2Stage_Vertical_Cutting_P3;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class SWidthBased_PWidthBased_2D_2Stage_Vertical_Cutting_P3{
  public static void main(String[] args) {
    long startTime = System.currentTimeMillis();
    ArrayList<Double> sWidth = new ArrayList<Double>();
    ArrayList<Double> sHeight = new ArrayList<Double>();   // Unique sheet height.
    ArrayList<Double> nSheet = new ArrayList<Double>();   // Amount of sheet of
each size.

    ArrayList<Double> pWidth = new ArrayList<Double>();
    ArrayList<Double> pHeight = new ArrayList<Double>();
    ArrayList<Double> nPanel = new ArrayList<Double>();

    ArrayList<Double> sLeftOverArea = new ArrayList<Double>();
    String excelFile = "C:\\Users\\Tiengkimseng\\Google
Drive\\Kimseng\\Thesis\\CODE_2D_2S_AND_2D_3S_CUTTING_8MODELS\\Heu
ristic_for_2D_2Stage_Con1_Con2_P1_P3\\SWidth_PWidth_P3\\CuttingData -
Sample 39.xlsx";
    //String excelFile = "D:\\Input and output of heuristics For construction 1 and
2.xlsx";
    System.out.println("ArrayList in row and column set: ");
    try {
      FileInputStream file = new FileInputStream(new File(excelFile));
      XSSFWorkbook workbook = new XSSFWorkbook(file);
      //XSSFSheet sheet = workbook.getSheet("Input4");// or .getsheetAt(2);

      XSSFSheet sheet = workbook.getSheet("Input");// or .getsheetAt(2);
      // cRow = currentRow, pRow = processingRow
      int cRow = 0 , pRow = 0;          // cRow can give any values but pRow cannot
(what is its function?)
      int cColumn = 1;           // here we also can change it
```

```
int dataInRow = 0;

//Iterate through each rows from first sheet
Iterator<Row> rowIterator = sheet.iterator();
while(rowIterator.hasNext()) {
  Row row = rowIterator.next();
  cRow++;
  cColumn = 1;
  dataInRow = 0;

  //For each row, iterate through each columns
  Iterator<Cell> cellIterator = row.cellIterator();
  while(cellIterator.hasNext()) {
    Cell cell = cellIterator.next();
    switch(cell.getCellType()) {
    case Cell.CELL_TYPE_BOOLEAN:
      System.out.print(cell.getBooleanCellValue() + "\t\t");
      dataInRow ++;
      break;
    case Cell.CELL_TYPE_NUMERIC:
      System.out.print(cell.getNumericCellValue() + "\t\t");
      if(cRow>pRow) {
        pRow = cRow;
      }
      Else
      {
        switch(cColumn) {
        case 1:
          pHeight.add(cell.getNumericCellValue());
          break;
        case 2:
          pWidth.add(cell.getNumericCellValue());
          break;
        case 3:
          nPanel.add(cell.getNumericCellValue());
          break;
        case 4:
          sHeight.add(cell.getNumericCellValue());
          break;
        case 5:
          sWidth.add(cell.getNumericCellValue());
          break;
        case 6:
          nSheet.add(cell.getNumericCellValue());
        }
        cColumn++;
      }
```

```
            dataInRow++;
            break;
          case Cell.CELL_TYPE_STRING:
            System.out.print(cell.getStringCellValue() + "\t\t");
            dataInRow++;
            break;
          }
        }
        if(dataInRow > 0) {
          System.out.println("");
        }
      }
      file.close();
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }

    ArrayList<Double> sheetWidth = new ArrayList<Double>();
    ArrayList<Double> sheetHeight = new ArrayList<Double>();

    ArrayList<Double> panelWidth = new ArrayList<Double>();
    ArrayList<Double> panelHeight = new ArrayList<Double>();
    ArrayList<Double> totalWastePerSheet = new ArrayList<Double>();

    //define sheet set
    System.out.println("\n" + "Sheet set: ");
    for (int i=0; i<nSheet.size(); i++) {
      for (int a=0; a<nSheet.get(i); a++) {
        sheetWidth.add(sWidth.get(i));
        sheetHeight.add(sHeight.get(i));
        sLeftOverArea.add(sWidth.get(i) * sHeight.get(i));
        //System.out.print( "[" + sHeight.get(i) + " x " + sWidth.get(i) + "]" + ", ");//It is
correct
      }System.out.println();
    }
    //define panel set
    System.out.println("\n" + "Panel set:");
    for (int i=0; i<nPanel.size(); i++) {
      for (int a=0; a<nPanel.get(i); a++) {
        panelWidth.add(pWidth.get(i));
        panelHeight.add(pHeight.get(i));
        //System.out.print( "[" + pHeight.get(i) + " x " + pWidth.get(i)+ "]" + ", ");//It is
correct
      }System.out.println();
    }
```

```
    //here is the process of cutting
    try {
      // Open excel file.
      FileInputStream file = new FileInputStream(new File(excelFile));

      XSSFWorkbook workbook = new XSSFWorkbook(file);
      XSSFSheet sheet = workbook.getSheet("Output");// or .getsheetAt(2);

      Row row; // Declare row variable as Excel row.
      Cell cell;  // Declare cell variable as Excel cell.
      System.out.println("\n" + "Here is the step of cutting: ");

      double leftoverHeight;
      double leftoverWidth;
      double wasteBeforeCut = 0;
      // Now everything is still working as expected. So no other code editing is
necessary.
      int startRow = 2, sHeightCell = 0, sWidthCell = 1, pHeightCell = 2, pWidthCell
= 3, levelCell = 4, sLeftOverAreaCell = 5, computeTimeCell = 6, totalWasteCell = 7;
      int r=0;
      long computeTime = 0;
      boolean existingLevel = false;  // Level check if new calculate new
[leftoverHeight] and [leftoverWidth].
      int i=0, j=0, level;

      for (i = 0; i < sheetWidth.size(); i++) {
        leftoverHeight = sheetHeight.get(i);
        leftoverWidth = sheetWidth.get(i);//.get(i) use in arraylist
        level = 0;

        row = sheet.createRow(startRow + r);

        cell = row.createCell(sHeightCell);
        cell.setCellValue(sheetHeight.get(i));

        cell = row.createCell(sWidthCell);
        cell.setCellValue(sheetWidth.get(i));

        //To define the value of the waste
        if (i > 0) {
          sheet.getRow(startRow + (r-
1)).createCell(sLeftOverAreaCell).setCellValue(sLeftOverArea.get(i-1));
        }
        //this loop use to define each level in a specific sheet
        for (double h=leftoverWidth; h>=panelWidth.get(panelWidth.size()-1);
h=leftoverWidth) {
          leftoverHeight = sheetHeight.get(i);
```

```
        existingLevel = false;
        // this loop use to pick the panel to put inside each level in a specific sheet
        for (j = 0; j < panelWidth.size(); j++) {
          if (existingLevel == false && (leftoverHeight>=panelHeight.get(j) &&
leftoverWidth>=panelWidth.get(j))) {
            leftoverWidth = leftoverWidth - panelWidth.get(j);
            existingLevel = true;
            level += 1;
          }
          if (existingLevel == true && leftoverHeight>=panelHeight.get(j)) {
            System.out.println("put panel: " + panelHeight.get(j) + " x " +
panelWidth.get(j) + " into sheet: " + (i+1) + " level: " + level);
            sLeftOverArea.set(i, sLeftOverArea.get(i) - (panelHeight.get(j) *
panelWidth.get(j)));
            //row = sheet.getRow(startRow + r);        // read data at row: startRow + r
in sheet in Excel file.
            try {
              row = sheet.getRow(startRow + r);
              if (row == null) {
                row = sheet.createRow(startRow + r);
              }
            }
            catch (Exception e) {
              row = sheet.createRow(startRow + r);
            }
            row.createCell(pHeightCell).setCellValue(panelHeight.get(j));
            row.createCell(pWidthCell).setCellValue(panelWidth.get(j));
            row.createCell(levelCell).setCellValue(level);

            r++;// move to new row.
            leftoverHeight = leftoverHeight - panelHeight.get(j);
            panelHeight.remove(j);
            panelWidth.remove(j);
            j = j - 1;
          }
        }
        if(panelHeight.isEmpty()){
          sheet.getRow(startRow + (r-
1)).createCell(sLeftOverAreaCell).setCellValue(sLeftOverArea.get(i));
          sheet.getRow(startRow + (r-
1)).createCell(computeTimeCell).setCellValue(computeTime);
          totalWastePerSheet.add(sLeftOverArea.get(i));

          for (int k = 0; k < totalWastePerSheet.size(); k++) {
            wasteBeforeCut = wasteBeforeCut + totalWastePerSheet.get(k);
          }
```

```
            sheet.getRow(startRow + (r-
1)).createCell(totalWasteCell).setCellValue(wasteBeforeCut);
            file.close();
            FileOutputStream outFile =new FileOutputStream(new File(excelFile));
            workbook.write(outFile);
            outFile.close();
            return;
          }
        }
        totalWastePerSheet.add(sLeftOverArea.get(i));
        sheetHeight.remove(0);
        sheetWidth.remove(0);
        long endTime = System.currentTimeMillis();
        computeTime = endTime - startTime;
        System.out.println("Time taken for this process are: " + (computeTime) + "
milli seconds" );
        cell = row.createCell(computeTimeCell);
        cell.setCellValue(computeTime);
      }
      sheet.getRow(startRow + (r-
1)).createCell(sLeftOverAreaCell).setCellValue(sLeftOverArea.get(9));
      file.close();
      FileOutputStream outFile =new FileOutputStream(new File(excelFile));
      workbook.write(outFile);
      outFile.close();

      for(int l=0; l < sLeftOverArea.size(); l++) {
        System.out.println("Sheet: " + (l+1) + " Left over area: " +
sLeftOverArea.get(l));
      }
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

7.1.8. Minimum Sheet Width Ordering Panel Height Horizontal Cut

```
package Simple_Heuristic_II_2D_2S_P3;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class Im1_Min_SWidth_PH_Cut_Horizon_P3{
  public static void main(String[] args) {
    long startTime = System.currentTimeMillis();
    ArrayList<Double> sWidth = new ArrayList<Double>();
    ArrayList<Double> sHeight = new ArrayList<Double>();   // Unique sheet height.
    ArrayList<Double> nSheet = new ArrayList<Double>();   // Amount of sheet of
each size.

    ArrayList<Double> pWidth = new ArrayList<Double>();
    ArrayList<Double> pHeight = new ArrayList<Double>();
    ArrayList<Double> nPanel = new ArrayList<Double>();

    ArrayList<Double> sLeftOverArea = new ArrayList<Double>();
    String excelFile = "C:\\Users\\Tiengkimseng\\Google
Drive\\Kimseng\\Thesis\\CODE_2D_2S_AND_2D_3S_CUTTING_8MODELS\\Sim
ple Heuristic II_VS_Coordinate
(x,y)_MinSWidth_P3\\SWidth_PHeight_P3\\CuttingData - Sample 28.xlsx";

    System.out.println("ArrayList in row and column set: ");
    ReadExcel(sWidth, sHeight, nSheet, pWidth, pHeight, nPanel, excelFile);

    ArrayList<Double> sheetWidth = new ArrayList<Double>();
    ArrayList<Double> sheetHeight = new ArrayList<Double>();

    ArrayList<Double> panelWidth = new ArrayList<Double>();
    ArrayList<Double> panelHeight = new ArrayList<Double>();

    ArrayList<Double> overCutPanel = new ArrayList<Double>();
    ArrayList<Double> overUnderCutPanel = new ArrayList<Double>();
```

```
    ArrayList<Double> wasteForEachOverCutPanelForEachSheet = new
ArrayList<Double>();
    ArrayList<Double> totalWastePerSheetArray = new ArrayList<Double>();

    //define sheet set
    System.out.println("\n" + "Sheet set: ");
    for (int i=0; i<nSheet.size(); i++) {
      for (int a=0; a<nSheet.get(i); a++) {
        sheetWidth.add(sWidth.get(i));
        sheetHeight.add(sHeight.get(i));
        sLeftOverArea.add(sWidth.get(i) * sHeight.get(i));
        System.out.print( "[" + sHeight.get(i) + " x " + sWidth.get(i) + "]" + ", ");
      }System.out.println();
    }
    //define panel set
    System.out.println("\n" + "Panel set:");
    for (int i=0; i<nPanel.size(); i++) {
      for (int a=0; a<nPanel.get(i); a++) {
        panelWidth.add(pWidth.get(i));
        panelHeight.add(pHeight.get(i));
        System.out.print( "[" + pHeight.get(i) + " x " + pWidth.get(i)+ "]" + ", ");
      }System.out.println();
    }
    double maxSHeight = 0;
    double maxSWidth = 0;
    for (int j = 0; j < nPanel.size(); j++) {
      for (int i = 0; i < nSheet.size(); i++) {
        if (sHeight.get(i) > maxSHeight) {
          maxSHeight = sHeight.get(i);
        }
        if (sWidth.get(i) > maxSWidth) {
          maxSWidth = sWidth.get(i);
        }
      }
      if (((pHeight.get(j) > maxSHeight)) || (pWidth.get(j) > maxSWidth)) {
        pWidth.remove(j);
        pHeight.remove(j);
        nPanel.remove(j);
        j--;
      }
    }
    //To check and remove the small sheet that we cannot use to cut other panel
    double minPHeight = 999999999;
    double minPWidth = 999999999;
    for (int i = 0; i < nSheet.size(); i++) {
      for (int j = 0; j < nPanel.size(); j++) {
```

```
        if (minPHeight > pHeight.get(j)) {
          minPHeight = pHeight.get(j);
        }
        if (minPWidth > pWidth.get(j)) {
          minPWidth = pWidth.get(j);
        }
      }
      if ((sHeight.get(i) < minPHeight) || (sWidth.get(i) < minPWidth)) {
        sWidth.remove(i);
        sHeight.remove(i);
        nSheet.remove(i);
        i--;
      }
    }
    //here is the process of cutting
    try {
      // Open excel file.
      FileInputStream file = new FileInputStream(new File(excelFile));

      //Get the workbook instance for XLSX file
      XSSFWorkbook workbook = new XSSFWorkbook(file);

      //Read sheet['Output']
      XSSFSheet sheet = workbook.getSheet("Output");

      Row row; // Declare row variable as Excel row.
      Cell cell;  // Declare cell variable as Excel cell.

      System.out.println("\n" + "Here is the step of cutting: ");

      ArrayList<Integer> chosenPanelIndex = new ArrayList<Integer>();
      ArrayList<Integer> chosenPanelAmount = new ArrayList<Integer>();

      ArrayList<Integer> usedPanelIndex = new ArrayList<Integer>();//the index of
the panel that have been cut in each type of sheet
      ArrayList<Integer> usedPanelAmount = new ArrayList<Integer>();//number of
panel in each type that used to cut
      ArrayList<Double> leftOverDemand = new ArrayList<Double>();//number of
demand after update with the panel number cut
      ArrayList<Double> xOrdinateArray = new ArrayList<Double>();//Use to add the
x ordinate one by one then when the panel is cut then clone it to xOrdinateSelected
when the sheet is full
      ArrayList<Double> yOrdinateArray = new ArrayList<Double>();
      ArrayList<Double> xOrdinateSelected = new ArrayList<Double>();//use to write
the coordinate in excel
      ArrayList<Double> yOrdinateSelected = new ArrayList<Double>();
```

```
        ArrayList<Double> pHeightArray = new ArrayList<Double>();//Use to add the
panel cut in order one by one then clone it to pHeightSelected when the sheet cut is
full
        ArrayList<Double> pWidthArray = new ArrayList<Double>();

        ArrayList<Double> pHeightSelected = new ArrayList<Double>();//use to write
the panel cut in excel in accordance with the coordinate
        ArrayList<Double> pWidthSelected = new ArrayList<Double>();

        ArrayList<Double> panelHeightAtBeginningLevel = new ArrayList<Double>();
        ArrayList<Double> panelHeightAtBeginningLevelClone = new
ArrayList<Double>();

        ArrayList<Double> indexOfMinWidthGapCopy = new ArrayList<Double>();
        ArrayList<Double> indexOfMinWidthGapCopyClone = new
ArrayList<Double>();

        double leftoverHeight;
        double leftoverWidth;
        boolean existingLevel = false;  // New or existing level check
        double currentLeftOverArea=0;
        double smallestLeftOverArea=0;          // Variable to store the smallest left over
area of sheet.
        int smallestLeftOverSheet=0;     // Store index of the smallest left over area sheet.
        int numberPanelPerSheet=0;
        int panelAmountWithMinimumWaste=0;
        long endTime = 0;
        double xOrdinate = 0;
        double yOrdinate = 0;
        double w = 0;
        int startRow = 2, numberingCell = 0, sHeightCell = 1, sWidthCell = 2,
pHeightCell = 3, pWidthCell = 4, demandCell = 5, wastePerSheetCell = 6,
numberPanelPerSheetCell = 7, numberSheetNeededToCut = 8, leftOverPanelCell = 9,
computeTimeCell = 10, xCell = 11, yCell = 12,       overCutPanelCell = 13,
totalWastePerSheetCell = 14, wasteForCuttingCell = 15;
        int r=0;
        int printNo = 0;
        int i=0, j=0, k=0, level = 0;
        int count = 0;
        double totalWastePerSheet = 0;
        int indexOfMinWidthGap = 0;
        //this loop for pick sheet one by one from all sheet
        for (j = 0; j < nPanel.size(); j++) {
          smallestLeftOverArea = sWidth.get(0)*sHeight.get(0);
          row = sheet.createRow(startRow + r);
          for  (i = 0; i < sHeight.size(); i++) {
            numberPanelPerSheet = 0;
```

```
        leftoverHeight = sHeight.get(i);
        level = 0;
        System.out.print("\n" + "Sheet: " + (i+1) + " Size: " + sWidth.get(i) + " x " +
sHeight.get(i));
        currentLeftOverArea =  sWidth.get(i) * sHeight.get(i);
        //this loop use to define each level in a specific sheet
        for (double h=leftoverHeight; h>=pHeight.get(pHeight.size()-1);
h=leftoverHeight) {//pHeight.get(pHeight.size()-1)= the last position of the which is
equal to 3 where we start from 0,1,2,3
          leftoverWidth = sWidth.get(i);
          existingLevel = false;
          ArrayList<Double>newPWidth = new ArrayList<Double>();
          ArrayList<Double>newPHeight = new ArrayList<Double>();
          ArrayList<Double>newNPanel = new ArrayList<Double>();

          if (sHeight.get(i) == leftoverHeight) {//use to start the coordinate of y
            yOrdinate = 0;
          }
          panelHeightAtBeginningLevel.add(pHeight.get(0));// Just want to add the
first panel before other panel add in
          for (int l = 0; l < pHeight.size(); l++) {// use to continue to the next level by
increase y
            if (sHeight.get(i)!= leftoverHeight) {
              if (pHeight.get(l) <= leftoverHeight && pWidth.get(l) <= leftoverWidth)
{
                yOrdinate = yOrdinate + panelHeightAtBeginningLevelClone.get(1);
                break;
              }
              if (pHeight.get(l) > leftoverHeight){
                continue;
              }
            }
          }
          level += 1;
          System.out.println();
          System.out.print("\tLevel: " + level + " |");
          //to check the panel in case that it satisfies only one criteria like sample 34
          if (existingLevel == false && ((leftoverHeight>=pHeight.get(j) &&
leftoverWidth>=pWidth.get(j)) )){
            existingLevel = true;
          } else {
            newPWidth = (ArrayList)pWidth.clone();
            newPHeight =(ArrayList)pHeight.clone();
            newNPanel = (ArrayList)nPanel.clone();

            newPWidth.remove(pWidth.size()-1);
            newPHeight.remove(pHeight.size()-1);
```

```
            newNPanel.remove(nPanel.size()-1);
          }
          newPWidth.clear();
          newPHeight.clear();
          newNPanel.clear();
          panelHeightAtBeginningLevelClone.clear();
          ArrayList <Double> leftoverWidthGapArray = new ArrayList<Double>();
          //Start cutting for the first panel that put into the sheet
          if (leftoverHeight >= pHeight.get(j)) {
            indexOfMinWidthGap = 0;
            for ( w = leftoverWidth; w >= pWidth.get(indexOfMinWidthGap); w =
leftoverWidth) {
            // To do: make this loop take demand into consideration.
              if (existingLevel == true &&
leftoverWidth>=pWidth.get(indexOfMinWidthGap)) {
                System.out.print("  " + pWidth.get(indexOfMinWidthGap) + " x " +
pHeight.get(indexOfMinWidthGap) + "  |");
                if (sWidth.get(i) == leftoverWidth) {
                  pHeightArray.add(pHeight.get(indexOfMinWidthGap));
                  pWidthArray.add(pWidth.get(indexOfMinWidthGap));
                  panelHeightAtBeginningLevel.add(pHeight.get(indexOfMinWidthGap
));

                  xOrdinate = 0;
                  xOrdinateArray.add(xOrdinate);
                  yOrdinateArray.add(yOrdinate);
                } else {
                  pHeightArray.add(pHeight.get(indexOfMinWidthGap));
                  pWidthArray.add(pWidth.get(indexOfMinWidthGap));
                  xOrdinate = xOrdinate + pWidthArray.get(pWidthArray.size()-2);
                  xOrdinateArray.add(xOrdinate);
                  yOrdinateArray.add(yOrdinate);
                }
                leftoverWidth = leftoverWidth - pWidth.get(indexOfMinWidthGap);
                currentLeftOverArea = currentLeftOverArea-
(pWidth.get(indexOfMinWidthGap)*pHeight.get(indexOfMinWidthGap));
                numberPanelPerSheet += 1;
                if(usedPanelIndex.indexOf(indexOfMinWidthGap) == -1) {
                  usedPanelIndex.add(indexOfMinWidthGap);
                  usedPanelAmount.add(1);
                } else {
                  usedPanelAmount.set(usedPanelIndex.indexOf(indexOfMinWidthGap)
, usedPanelAmount.get(usedPanelIndex.indexOf(indexOfMinWidthGap))+1);//use to
increase the amount of the same panel size that put into the sheet
                }
                indexOfMinWidthGapCopy.add((double) indexOfMinWidthGap);
```

```
                indexOfMinWidthGapCopyClone = (ArrayList<Double>)
indexOfMinWidthGapCopy.clone();
              }
              //To select the panel that can give the minimum of leftoverWidth % for all
pWidth
              double minWidthGap = 900000000;
              for (int l = 0; l < pWidth.size(); l++) {
                double leftoverWidthGap = leftoverWidth % pWidth.get(l);
                leftoverWidthGapArray.add(leftoverWidthGap);
                if (l >= indexOfMinWidthGapCopyClone.get(0)) {
                  if (leftoverWidthGap < minWidthGap) {
                    minWidthGap = leftoverWidthGap;
                    indexOfMinWidthGap =
leftoverWidthGapArray.indexOf(minWidthGap);
                  }
                }
                leftoverWidthGapArray.set(l, 100000.0);
              }
              leftoverWidthGapArray.clear();
              indexOfMinWidthGapCopyClone.clear();
            }
          }
          //Start cutting for the next panel that put into the sheet when we put other
types of panel in the same level or the next level
          ArrayList<Double> nextPanelCut = new ArrayList<Double>();
          for ( k = (indexOfMinWidthGap+1); k < pWidth.size(); k++) {
            if(pHeight.get(k) <= leftoverHeight && pWidth.get(k) <= leftoverWidth) {
              nextPanelCut.add((double) k);// want to pick the first panel to calculate
the leftoverHeight
              //to pick the new coordinate in the new level
              if (sWidth.get(i) == leftoverWidth) {
                pHeightArray.add(pHeight.get(k));
                panelHeightAtBeginningLevel.add(pHeight.get(k));
                pWidthArray.add(pWidth.get(k));

                xOrdinate = 0;
                xOrdinateArray.add(xOrdinate);
                yOrdinateArray.add(yOrdinate);
              } else {
                pHeightArray.add(pHeight.get(k));
                pWidthArray.add(pWidth.get(k));
                xOrdinate = xOrdinate + pWidthArray.get(pWidthArray.size()-2);

                xOrdinateArray.add(xOrdinate);
                yOrdinateArray.add(yOrdinate);
              }
              leftoverWidth -= pWidth.get(k);
```

```
                currentLeftOverArea -= (pWidth.get(k)*pHeight.get(k));
                System.out.print("  " + pWidth.get(k) + " x " + pHeight.get(k) + "  |");
                numberPanelPerSheet += 1;
                if(usedPanelIndex.indexOf(k) == -1) {
                  usedPanelIndex.add(k);
                  usedPanelAmount.add(1);
                } else {
                  usedPanelAmount.set(usedPanelIndex.indexOf(k),
usedPanelAmount.get(usedPanelIndex.indexOf(k))+1);
                }
                indexOfMinWidthGapCopy.add((double) k);
                indexOfMinWidthGapCopyClone = (ArrayList<Double>)
indexOfMinWidthGapCopy.clone();
              //To select the panel that can give the minimum of leftoverWidth % for all
pWidth
                double minWidthGap = 900000000;
                for (int l = 0; l < pWidth.size(); l++) {
                  double leftoverWidthGap = leftoverWidth % pWidth.get(l);
                  leftoverWidthGapArray.add(leftoverWidthGap);
                  if (l >= indexOfMinWidthGapCopyClone.get(0)) {
                  // Pick only the smaller panel to put into each level
                    if (leftoverWidthGap < minWidthGap) {
                      minWidthGap = leftoverWidthGap;//Find the panel to put that can
give the minimum leftover width
                      indexOfMinWidthGap =
leftoverWidthGapArray.indexOf(minWidthGap);
                    }
                  }
                  leftoverWidthGapArray.set(l, 100000.0);//to avoid picking the first panel
when the result of % are the same
                }
                leftoverWidthGapArray.clear();
                indexOfMinWidthGapCopyClone.clear();
                k = indexOfMinWidthGap - 1 ;
              }
            }
            panelHeightAtBeginningLevelClone =
(ArrayList)panelHeightAtBeginningLevel.clone();
            panelHeightAtBeginningLevel.clear();
            if (leftoverHeight >= pHeight.get(j)) {
              leftoverHeight = leftoverHeight - pHeight.get(j);
            } else {
              if (nextPanelCut.size()!=0) {
                double a = nextPanelCut.get(0);
                leftoverHeight = leftoverHeight - pHeight.get((int) a);
              }
              if (nextPanelCut.isEmpty()) {
```

```
                leftoverHeight = 0;
                continue;
              }
            }
            indexOfMinWidthGapCopy.clear();
          }
          System.out.println();
          System.out.print("Total Waste for each sheet: " + currentLeftOverArea);
          System.out.print("\nPanel fit: " + numberPanelPerSheet);
          System.out.println();

          if (smallestLeftOverArea > currentLeftOverArea) {
            smallestLeftOverArea = currentLeftOverArea;
            smallestLeftOverSheet = i;
            panelAmountWithMinimumWaste = numberPanelPerSheet;
            //We have to copy it to use when we write the answer in excel
            pHeightSelected = (ArrayList)pHeightArray.clone();
            pWidthSelected = (ArrayList)pWidthArray.clone();

            xOrdinateSelected = (ArrayList)xOrdinateArray.clone();
            yOrdinateSelected = (ArrayList)yOrdinateArray.clone();

            chosenPanelIndex = (ArrayList)usedPanelIndex.clone();
            chosenPanelAmount = (ArrayList)usedPanelAmount.clone();
          }
          for(int ind =0; ind < chosenPanelIndex.size(); ind++) {
            System.out.println("Used Index " + chosenPanelIndex.get(ind));
            System.out.println("Used Amount " + chosenPanelAmount.get(ind));
          }
          // clear it first before selecting the next panel to cut; otherwise, it adds on the
existing data
          pHeightArray.clear();
          pWidthArray.clear();

          xOrdinateArray.clear();
          yOrdinateArray.clear();

          usedPanelIndex.clear();
          usedPanelAmount.clear();
        }
        System.out.println();
        System.out.println(panelAmountWithMinimumWaste + " panels are cut from
panel set by putting in sheet number " + (smallestLeftOverSheet+1) + " (" +
sWidth.get(smallestLeftOverSheet) + " x " + sHeight.get(smallestLeftOverSheet) + ")
with the minimum waste " + smallestLeftOverArea);
```

```
        double minimumSheet =
10000000;//Math.ceil(nPanel.get(chosenPanelIndex.get(0)) /
chosenPanelAmount.get(0))
        double result;
        // This loop use to select the pattern that can give the minimum waste to cut.
        for(int d=0; d < chosenPanelIndex.size(); d++) {
          result = Math.ceil(nPanel.get(chosenPanelIndex.get(d)) /
chosenPanelAmount.get(d));
          if (minimumSheet > result) {
            minimumSheet = result;
          }
          System.out.println("To fulfill demand of " +
nPanel.get(chosenPanelIndex.get(d)) + " we need to use " + result + " sheets");
        }
        System.out.println("\nTherefore: To satisfy the demand we need: " +
minimumSheet + " sheets.");
        //To find the over cut panel
        double totalWasteOfOverCutPanel = 0;
        for (int l = 0; l < chosenPanelIndex.size(); l++) {
          overUnderCutPanel.add((minimumSheet * chosenPanelAmount.get(l)) -
nPanel.get(chosenPanelIndex.get(l)));
          if (overUnderCutPanel.get(l)<= 0) {
            overCutPanel.add(0.0);
          } else {
            overCutPanel.add(overUnderCutPanel.get(l));
          }
        }//To cut the panel based on the index of the panel selected
        for (int l = 0; l < chosenPanelIndex.size(); l++) {
          wasteForEachOverCutPanelForEachSheet.add(overCutPanel.get(l)*
pHeight.get(chosenPanelIndex.get(l)) * pWidth.get(chosenPanelIndex.get(l)));
          totalWasteOfOverCutPanel = totalWasteOfOverCutPanel +
wasteForEachOverCutPanelForEachSheet.get(l);//To find the waste of the over cut
panel
          System.out.println("Over cut Panel = " + overCutPanel);
        }
        totalWastePerSheet = (smallestLeftOverArea * minimumSheet) +
totalWasteOfOverCutPanel;//Total waste per sheet included the over cut panel
        totalWastePerSheetArray.add(totalWastePerSheet);

        int index;
        double amount;
        double demand;
        //Writing the panel of the pattern that we have selected to cut
        for(int ind = 0; ind < chosenPanelIndex.size(); ind++) {
          System.out.println("Chosen Index " + chosenPanelIndex.get(ind));
          System.out.println("Used Amount " + chosenPanelAmount.get(ind));
          row = sheet.createRow(startRow + r);
```

```
index = chosenPanelIndex.get(ind);
amount = chosenPanelAmount.get(ind);
demand = nPanel.get(chosenPanelIndex.get(ind));
leftOverDemand.add(demand - (amount * minimumSheet));
count = 0;
//Write only that first stage of cutting like No
if (ind == 0) {//ind = chosenPanelIndex
  cell = row.createCell(numberingCell);
  cell.setCellValue(printNo+1);

  cell = row.createCell(sHeightCell);
  cell.setCellValue(sHeight.get(smallestLeftOverSheet));

  cell = row.createCell(sWidthCell);
  cell.setCellValue(sWidth.get(smallestLeftOverSheet));

  if (leftOverDemand.get(ind)> 0) {
    cell = row.createCell(leftOverPanelCell);
    cell.setCellValue(leftOverDemand.get(ind));
  }
 }
 if (ind < chosenPanelIndex.size()-1) { //do not let it increase the row when it is
the last panel that we put coz we have to put other data on the last row
  cell = row.createCell(demandCell);
  cell.setCellValue(nPanel.get(chosenPanelIndex.get(ind)));

  cell = row.createCell(overCutPanelCell);
  cell.setCellValue(overCutPanel.get(ind));

  cell = row.createCell(numberPanelPerSheetCell);
  cell.setCellValue(chosenPanelAmount.get(ind));

  if (leftOverDemand.get(ind)> 0) {
    cell = row.createCell(leftOverPanelCell);
    cell.setCellValue(leftOverDemand.get(ind));
  }
  // to spread the panel of cutting such that i can put the coordinate
  for (int l = 0; l < chosenPanelAmount.get(ind); l++) {
    cell = row.createCell(pHeightCell);
    //cell.setCellValue(pHeight.get(chosenPanelIndex.get(ind)));

    cell.setCellValue(pHeightSelected.get(l));
    cell = row.createCell(pWidthCell);
    //cell.setCellValue(pWidth.get(chosenPanelIndex.get(ind)));

    cell.setCellValue(pWidthSelected.get(l));
    cell = row.createCell(xCell);
```

```
    cell.setCellValue(xOrdinateSelected.get(l));

    cell = row.createCell(yCell);
    cell.setCellValue(yOrdinateSelected.get(l));

    count = count + 1;
    r++;
    row = sheet.createRow(startRow + r);
   }
   //Remove that panel use such that it can print of other panel
   for (int b = (count-1) ; b >= 0; b--) {
    pHeightSelected.remove(b);
    pWidthSelected.remove(b);

    xOrdinateSelected.remove(b);
    yOrdinateSelected.remove(b);
   }
  } else if ((chosenPanelIndex.size()-1) == ind) {
   cell = row.createCell(demandCell);
   cell.setCellValue(nPanel.get(chosenPanelIndex.get(ind)));

   cell = row.createCell(overCutPanelCell);
   cell.setCellValue(overCutPanel.get(ind));

   cell = row.createCell(numberPanelPerSheetCell);
   cell.setCellValue(chosenPanelAmount.get(ind));

   if (leftOverDemand.get(ind)> 0) {
    cell = row.createCell(leftOverPanelCell);
    cell.setCellValue(leftOverDemand.get(ind));
   }
   for (int l = 0; l < chosenPanelAmount.get(ind); l++) {
    cell = row.createCell(pHeightCell);
    //cell.setCellValue(pHeight.get(chosenPanelIndex.get(ind)));
    cell.setCellValue(pHeightSelected.get(l));

    cell = row.createCell(pWidthCell);
    //cell.setCellValue(pWidth.get(chosenPanelIndex.get(ind)));
    cell.setCellValue(pWidthSelected.get(l));

    cell = row.createCell(xCell);
    cell.setCellValue(xOrdinateSelected.get(l));

    cell = row.createCell(yCell);
    cell.setCellValue(yOrdinateSelected.get(l));

    count = count + 1;
```

```
            r++;
            row = sheet.createRow(startRow + r);
          }
          for (int b = (count-1) ; b >= 0; b--) {
            pHeightSelected.remove(b);
            pWidthSelected.remove(b);

            xOrdinateSelected.remove(b);
            yOrdinateSelected.remove(b);
          }
          r--;
        }
      }
      wasteForEachOverCutPanelForEachSheet.clear();
      overUnderCutPanel.clear();
      overCutPanel.clear();
      r = r + 2;
      printNo++;
      cell = row.createCell(wastePerSheetCell);
      cell.setCellValue(smallestLeftOverArea);

      cell = row.createCell(numberSheetNeededToCut);
      cell.setCellValue(minimumSheet);

      cell = row.createCell(totalWastePerSheetCell);
      cell.setCellValue(totalWastePerSheet);

      endTime = System.currentTimeMillis();
      System.out.println("Time taken for this process are: " + (endTime - startTime) +
" milli seconds" );

      cell = row.createCell(computeTimeCell);
      cell.setCellValue(endTime - startTime);

      for( int d = chosenPanelIndex.size()-1; d >= 0; d--) {
        index = chosenPanelIndex.indexOf(Collections.max(chosenPanelIndex));
        amount = chosenPanelAmount.get(index);
        demand = nPanel.get(chosenPanelIndex.get(index));
        int newIndex = chosenPanelIndex.get(index);
        if (leftOverDemand.get(index) > 0) {
          nPanel.set(newIndex, leftOverDemand.get(index));
          chosenPanelIndex.remove(index);
          chosenPanelAmount.remove(index);
          leftOverDemand.remove(index);
        } else {
          nPanel.remove(newIndex);
          pHeight.remove(newIndex);
```

```
            pWidth.remove(newIndex);
            chosenPanelIndex.remove(index);
            chosenPanelAmount.remove(index);
            leftOverDemand.remove(index);
          }
        }
        leftOverDemand.clear();
        System.out.println("------------------------------------------------------------------
-------------------------");
        j--;
        double wasteForCutting = 0;
        for (int l = 0; l < totalWastePerSheetArray.size(); l++) {
          wasteForCutting = wasteForCutting + totalWastePerSheetArray.get(l);
        }
        cell = row.createCell(wasteForCuttingCell);
        cell.setCellValue(wasteForCutting);
      }
      chosenPanelIndex.clear();
      chosenPanelAmount.clear();
      file.close();

      FileOutputStream outFile =new FileOutputStream(new File(excelFile));
      workbook.write(outFile);
      outFile.close();
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
  public static void ReadExcel(ArrayList<Double> sWidth,
                               ArrayList<Double> sHeight, ArrayList<Double> nSheet,
                               ArrayList<Double> pWidth, ArrayList<Double> pHeight,
                               ArrayList<Double> nPanel, String excelFile) {
    try {
      FileInputStream file = new FileInputStream(new File(excelFile));
      //Get the workbook instance for XLSX file
      XSSFWorkbook workbook = new XSSFWorkbook(file);

      //Get first sheet from the workbook
      XSSFSheet sheet = workbook.getSheet("Input");// or .getsheetAt(2);

      // cRow = currentRow, pRow = processingRow
      int cRow = 0 , pRow = 0;          // cRow can give any values but pRow cannot
      int cColumn = 1;            // here we also can change it
      int dataInRow = 0;
```

```
//Iterate through each rows from first sheet
Iterator<Row> rowIterator = sheet.iterator();
while(rowIterator.hasNext()) {
  Row row = rowIterator.next();
  cRow++;
  cColumn = 1;
  dataInRow = 0;

  //For each row, iterate through each columns
  Iterator<Cell> cellIterator = row.cellIterator();
  while(cellIterator.hasNext()) {
    Cell cell = cellIterator.next();
    switch(cell.getCellType()) {
    case Cell.CELL_TYPE_BOOLEAN:
      System.out.print(cell.getBooleanCellValue() + "\t\t");
      dataInRow ++;
      break;
    case Cell.CELL_TYPE_NUMERIC:
      System.out.print(cell.getNumericCellValue() + "\t\t");
      if(cRow>pRow) {
        pRow = cRow;
      }
      Else
      {
        switch(cColumn) {
        case 1:
          pHeight.add(cell.getNumericCellValue());
          break;
        case 2:
          pWidth.add(cell.getNumericCellValue());
          break;
        case 3:
          nPanel.add(cell.getNumericCellValue());
          break;
        case 4:
          sHeight.add(cell.getNumericCellValue());
          break;
        case 5:
          sWidth.add(cell.getNumericCellValue());
          break;
        case 6:
          nSheet.add(cell.getNumericCellValue());
        }
        cColumn++;
      }
      dataInRow++;
      break;
```

```
            case Cell.CELL_TYPE_STRING:
              System.out.print(cell.getStringCellValue() + "\t\t");
              dataInRow++;
              break;
            }
          }
          if(dataInRow > 0) {
            System.out.println("");
          }
        }
        file.close();
      } catch (FileNotFoundException e) {
        e.printStackTrace();
      } catch (IOException e) {
        e.printStackTrace();
      }
    }
}
```

7.1.9. Minimum Sheet Height Ordering Panel Width Vertical Cut

```
package Simple_Heuristic_II_2D_2S_P3;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
public class Im2_Min_SHeight_PW_Cut_Vertical_P3{
  public static void main(String[] args) {
    long startTime = System.currentTimeMillis();
    ArrayList<Double> sWidth = new ArrayList<Double>();
    ArrayList<Double> sHeight = new ArrayList<Double>();   // Unique sheet height.
    ArrayList<Double> nSheet = new ArrayList<Double>();   // Amount of sheet of
each size.

    ArrayList<Double> pWidth = new ArrayList<Double>();
    ArrayList<Double> pHeight = new ArrayList<Double>();
    ArrayList<Double> nPanel = new ArrayList<Double>();

    ArrayList<Double> sLeftOverArea = new ArrayList<Double>();
    String excelFile = "C:\\Users\\Tiengkimseng\\Google
Drive\\Kimseng\\Thesis\\CODE_2D_2S_AND_2D_3S_CUTTING_8MODELS\\Sim
ple Heuristic II_VS_Coordinate
(x,y)_MinSWidth_P3\\SWidth_PWidth_P3\\CuttingData - Sample 39.xlsx";

    System.out.println("ArrayList in row and column set: ");
    ReadExcel(sWidth, sHeight, nSheet, pWidth, pHeight, nPanel, excelFile);

    ArrayList<Double> sheetWidth = new ArrayList<Double>();
    ArrayList<Double> sheetHeight = new ArrayList<Double>();

    ArrayList<Double> panelWidth = new ArrayList<Double>();
    ArrayList<Double> panelHeight = new ArrayList<Double>();

    ArrayList<Double> overCutPanel = new ArrayList<Double>();
    ArrayList<Double> overUnderCutPanel = new ArrayList<Double>();
```

```
    ArrayList<Double> wasteForEachOverCutPanelForEachSheet = new
ArrayList<Double>();
    ArrayList<Double> totalWastePerSheetArray = new ArrayList<Double>();

    //define sheet set
    System.out.println("\n" + "Sheet set: ");
    for (int i=0; i<nSheet.size(); i++) {
      for (int a=0; a<nSheet.get(i); a++) {
        sheetWidth.add(sWidth.get(i));
        sheetHeight.add(sHeight.get(i));
        sLeftOverArea.add(sWidth.get(i) * sHeight.get(i));
        //System.out.print( "[" + sHeight.get(i) + " x " + sWidth.get(i) + "]" + ", ");
      }System.out.println();
    }
    //define panel set
    System.out.println("\n" + "Panel set:");
    for (int i=0; i<nPanel.size(); i++) {
      for (int a=0; a<nPanel.get(i); a++) {
        panelWidth.add(pWidth.get(i));
        panelHeight.add(pHeight.get(i));
        //System.out.print( "[" + pHeight.get(i) + " x " + pWidth.get(i)+ "]" + ", ");
      }System.out.println();
    }
    // To check and remove the big panel that we cannot put into the sheet, if we don't
have it, it might cause to error.
    double maxSHeight = 0;
    double maxSWidth = 0;
    for (int j = 0; j < nPanel.size(); j++) {
      for (int i = 0; i < nSheet.size(); i++) {
        if (sHeight.get(i) > maxSHeight) {
          maxSHeight = sHeight.get(i);
        }
        if (sWidth.get(i) > maxSWidth) {
          maxSWidth = sWidth.get(i);
        }
      }
      if (((pHeight.get(j) > maxSHeight)) || (pWidth.get(j) > maxSWidth)) {
        pWidth.remove(j);
        pHeight.remove(j);
        nPanel.remove(j);
        j--;
      }
    }
    //To check and remove the small sheet that we cannot use to cut other panel, if we
don't have it, it might cause to error
    double minPHeight = 999999999;
    double minPWidth = 999999999;
```

```
    for (int i = 0; i < nSheet.size(); i++) {
      for (int j = 0; j < nPanel.size(); j++) {
        if (minPHeight > pHeight.get(j)) {
          minPHeight = pHeight.get(j);
        }
        if (minPWidth > pWidth.get(j)) {
          minPWidth = pWidth.get(j);
        }
      }
      if ((sHeight.get(i) < minPHeight) || (sWidth.get(i) < minPWidth)) {
        sWidth.remove(i);
        sHeight.remove(i);
        nSheet.remove(i);
        i--;
      }
    }
    //here is the process of cutting
    try {
      // Open excel file.
      FileInputStream file = new FileInputStream(new File(excelFile));
      //Get the workbook instance for XLSX file
      XSSFWorkbook workbook = new XSSFWorkbook(file);

      XSSFSheet sheet = workbook.getSheet("Output");
      Row row; // Declare row variable as Excel row.
      Cell cell;  // Declare cell variable as Excel cell.
      System.out.println("\n" + "Here is the step of cutting: ");

      ArrayList<Integer> chosenPanelIndex = new ArrayList<Integer>();
      ArrayList<Integer> chosenPanelAmount = new ArrayList<Integer>();

      ArrayList<Integer> usedPanelIndex = new ArrayList<Integer>();
      ArrayList<Integer> usedPanelAmount = new ArrayList<Integer>();
      ArrayList<Double> leftOverDemand = new ArrayList<Double>();

      ArrayList<Double> xOrdinateArray = new ArrayList<Double>();//Use to add the
x ordinate one by one then when the panel is cut then clone it to xOrdinateSelected
when the sheet is full
      ArrayList<Double> yOrdinateArray = new ArrayList<Double>();

      ArrayList<Double> xOrdinateSelected = new ArrayList<Double>();//use to write
the coordinate in excel
      ArrayList<Double> yOrdinateSelected = new ArrayList<Double>();
      ArrayList<Double> pHeightArray = new ArrayList<Double>();//Use to add the
panel cut in order one by one then clone it to pHeightSelected when the sheet cut is
full
      ArrayList<Double> pWidthArray = new ArrayList<Double>();
```

```
        ArrayList<Double> pHeightSelected = new ArrayList<Double>();//use to write
the panel cut in excel in accordance with the coordinate
        ArrayList<Double> pWidthSelected = new ArrayList<Double>();
        ArrayList<Double> panelWidthAtBeginningLevel = new ArrayList<Double>();
        ArrayList<Double> panelWidthAtBeginningLevelClone = new
ArrayList<Double>();

        ArrayList<Double> indexOfMinHeightGapCopy = new ArrayList<Double>();
        ArrayList<Double> indexOfMinHeightGapCopyClone = new
ArrayList<Double>();

        double leftoverHeight;
        double leftoverWidth;
        boolean existingLevel = false;  // New or existing level check
        double currentLeftOverArea=0;
        double smallestLeftOverArea=0;          // Variable to store the smallest left over
area of sheet.
        int smallestLeftOverSheet=0;     // Store index of the smallest left over area sheet.
        int numberPanelPerSheet=0;
        int panelAmountWithMinimumWaste=0;
        long endTime = 0;
        double xOrdinate = 0;
        double yOrdinate = 0;
        double w = 0;
        int startRow = 2, numberingCell = 0, sHeightCell = 1, sWidthCell = 2,
pHeightCell = 3, pWidthCell = 4, demandCell = 5, wastePerSheetCell = 6,
numberPanelPerSheetCell = 7, numberSheetNeededToCut = 8, leftOverPanelCell = 9,
computeTimeCell = 10, xCell = 11, yCell = 12, overCutPanelCell = 13,
totalWastePerSheetCell = 14, wasteForCuttingCell = 15;
        int r=0;
        int printNo = 0;
        int i=0, j=0, k=0, level = 0;
        int count = 0;
        double totalWastePerSheet = 0;
        int indexOfMinHeightGap = 0;
        //this loop for pick sheet one by one from all sheet
        for (j = 0; j < nPanel.size(); j++) {
          smallestLeftOverArea = sWidth.get(0)*sHeight.get(0);
          row = sheet.createRow(startRow + r);
          for  (i = 0; i < sHeight.size(); i++) {
            numberPanelPerSheet = 0;
            leftoverWidth = sWidth.get(i);
            level = 0;
            System.out.print("\n" + "Sheet: " + (i+1) + " Size: " + sWidth.get(i) + " x " +
sHeight.get(i));
            currentLeftOverArea =  sWidth.get(i) * sHeight.get(i);
            //this loop use to define each level in a specific sheet
```

```
        for (double h=leftoverWidth; h>=pWidth.get(pWidth.size()-1);
        h=leftoverWidth){
          //pHeight.get(pHeight.size()-1)= the last position of the which is equal to 3
where we start from 0,1,2,3
          leftoverHeight = sHeight.get(i);
          existingLevel = false;

          ArrayList<Double>newPWidth = new ArrayList<Double>();
          ArrayList<Double>newPHeight = new ArrayList<Double>();
          ArrayList<Double>newNPanel = new ArrayList<Double>();

          if (sWidth.get(i) == leftoverWidth) {//use to start the coordinate of y
            xOrdinate = 0;
          }
          panelWidthAtBeginningLevel.add(pWidth.get(0));// Just want to add the first
panel before other panel add in
          for (int l = 0; l < pWidth.size(); l++) {// use to continue to the next level by
increase y
            if (sWidth.get(i)!= leftoverWidth) {
              if (pWidth.get(l) <= leftoverWidth && pHeight.get(l) <= leftoverHeight)
{//the second part have just added to satisfy sample 34
                xOrdinate = xOrdinate + panelWidthAtBeginningLevelClone.get(1);
                break;
              }
              if (pWidth.get(l) > leftoverWidth){
                continue;

              }
            }
          }
          level += 1;
          System.out.println();
          System.out.print("\tLevel: " + level + " |");
          if (existingLevel == false && ((leftoverHeight>=pHeight.get(j) &&
leftoverWidth>=pWidth.get(j)) )){
            existingLevel = true;
          } else {//to check the panel in case that it satisfies only one criteria sample 34
            newPWidth = (ArrayList)pWidth.clone();
            newPHeight =(ArrayList)pHeight.clone();
            newNPanel = (ArrayList)nPanel.clone();

            newPWidth.remove(pWidth.size()-1);
            newPHeight.remove(pHeight.size()-1);
            newNPanel.remove(nPanel.size()-1);
          }
          newPWidth.clear();
          newPHeight.clear();
```

```
            newNPanel.clear();
            panelWidthAtBeginningLevelClone.clear();
            ArrayList <Double> leftoverHeightGapArray = new ArrayList<Double>();
            if (leftoverWidth >= pWidth.get(j)) {
              indexOfMinHeightGap = 0;
              for (w = leftoverHeight; w >= pHeight.get(indexOfMinHeightGap); w =
leftoverHeight) {
              // To do make this loop take demand into consideration.
                if (existingLevel == true && leftoverHeight >=
pHeight.get(indexOfMinHeightGap)) {
                  System.out.print("  " + pWidth.get(indexOfMinHeightGap) + " x " +
pHeight.get(indexOfMinHeightGap) + "  |");
                  if (sHeight.get(i) == leftoverHeight) {
                    pHeightArray.add(pHeight.get(indexOfMinHeightGap));
                    pWidthArray.add(pWidth.get(indexOfMinHeightGap));
                    panelWidthAtBeginningLevel.add(pWidth.get(indexOfMinHeightGap)
);
                    yOrdinate = 0;
                    xOrdinateArray.add(xOrdinate);
                    yOrdinateArray.add(yOrdinate);
                  } else {
                    pHeightArray.add(pHeight.get(indexOfMinHeightGap));
                    pWidthArray.add(pWidth.get(indexOfMinHeightGap));
                    yOrdinate = yOrdinate + pHeightArray.get(pHeightArray.size()-2);
                    xOrdinateArray.add(xOrdinate);
                    yOrdinateArray.add(yOrdinate);
                  }
                  leftoverHeight = leftoverHeight - pHeight.get(indexOfMinHeightGap);
                  currentLeftOverArea = currentLeftOverArea-
(pWidth.get(indexOfMinHeightGap)*pHeight.get(indexOfMinHeightGap));
                  numberPanelPerSheet += 1;
                  if(usedPanelIndex.indexOf(indexOfMinHeightGap) == -1) {
                    usedPanelIndex.add(indexOfMinHeightGap);
                    usedPanelAmount.add(1);
                  } else {
                    usedPanelAmount.set(usedPanelIndex.indexOf(indexOfMinHeightGap
), usedPanelAmount.get(usedPanelIndex.indexOf(indexOfMinHeightGap))+1);//use
to increase the amount of the same panel size that put into the sheet
                  }
                  indexOfMinHeightGapCopy.add((double) indexOfMinHeightGap);
                  indexOfMinHeightGapCopyClone = (ArrayList<Double>)
indexOfMinHeightGapCopy.clone();
                }
                //To select the panel that can give the minimum of leftoverWidth % for all
pWidth
                double minHeightGap = 900000000;
                for (int l = 0; l < pWidth.size(); l++) {
```

```
                double leftoverHeightGap = leftoverHeight % pHeight.get(l);
                leftoverHeightGapArray.add(leftoverHeightGap);
                if (l >= indexOfMinHeightGapCopyClone.get(0)) {
                  if (leftoverHeightGap < minHeightGap) {
                    minHeightGap = leftoverHeightGap;
                    indexOfMinHeightGap =
leftoverHeightGapArray.indexOf(minHeightGap);
                  }
                }
                leftoverHeightGapArray.set(l, 100000.0);
              }
              leftoverHeightGapArray.clear();
              indexOfMinHeightGapCopyClone.clear();
            }
          }
          ArrayList<Double> nextPanelCut = new ArrayList<Double>();
          for ( k = indexOfMinHeightGap+1; k < pHeight.size(); k++) {
            if(pHeight.get(k) <= leftoverHeight && pWidth.get(k) <= leftoverWidth) {
              nextPanelCut.add((double) k);
              // want to pick the first panel to calculate the leftoverHeight
              if (sHeight.get(i) == leftoverHeight) {
                pWidthArray.add(pWidth.get(k));
                panelWidthAtBeginningLevel.add(pWidth.get(k));
                pHeightArray.add(pHeight.get(k));
                yOrdinate = 0;
                xOrdinateArray.add(xOrdinate);
                yOrdinateArray.add(yOrdinate);
              } else {
                pHeightArray.add(pHeight.get(k));
                pWidthArray.add(pWidth.get(k));
                yOrdinate = yOrdinate + pHeightArray.get(pHeightArray.size()-2);
                //xOrdinate = xOrdinate + pWidth.get(j);
                xOrdinateArray.add(xOrdinate);
                yOrdinateArray.add(yOrdinate);
              }
              leftoverHeight -= pHeight.get(k);
              currentLeftOverArea -= (pWidth.get(k) * pHeight.get(k));
              System.out.print("  " + pWidth.get(k) + " x " + pHeight.get(k) + "  |");
              numberPanelPerSheet += 1;
              if(usedPanelIndex.indexOf(k) == -1) {
                usedPanelIndex.add(k);
                usedPanelAmount.add(1);
              } else {
                usedPanelAmount.set(usedPanelIndex.indexOf(k),usedPanelAmount.get
(usedPanelIndex.indexOf(k))+1);
              }
              indexOfMinHeightGapCopy.add((double) k);
```

```
            indexOfMinHeightGapCopyClone = (ArrayList<Double>)
indexOfMinHeightGapCopy.clone();
            //To select the panel that can give the minimum of leftoverWidth % for all
pWidth
            double minHeightGap = 900000000;
            for (int l = 0; l < pWidth.size(); l++) {
              double leftoverHeightGap = leftoverHeight % pHeight.get(l);
              leftoverHeightGapArray.add(leftoverHeightGap);
              if (l >= indexOfMinHeightGapCopyClone.get(0)) {// Pick only the
smaller panel to put into each level
                if (leftoverHeightGap < minHeightGap) {
                  minHeightGap = leftoverHeightGap;//Find the panel to put that can
give the minimum leftover width
                  indexOfMinHeightGap =
leftoverHeightGapArray.indexOf(minHeightGap);
                }
              }
              leftoverHeightGapArray.set(l, 100000.0);//to avoid picking the first
panel when the result of % are the same
            }
            leftoverHeightGapArray.clear();
            indexOfMinHeightGapCopyClone.clear();
            k = indexOfMinHeightGap - 1 ;
          }
        }
        panelWidthAtBeginningLevelClone =
(ArrayList)panelWidthAtBeginningLevel.clone();
        panelWidthAtBeginningLevel.clear();
        if (leftoverWidth >= pWidth.get(j)) {
          leftoverWidth = leftoverWidth - pWidth.get(j);
        } else {
          if (nextPanelCut.size()!= 0) {
            double a = nextPanelCut.get(0);//to find the leftoverWidth when the first
panel cannot fill in, and the next panel is fit.
            leftoverWidth = leftoverWidth - pWidth.get((int) a);// p in the index of
the panel of the other side
          }
          if (nextPanelCut.isEmpty()) {//In sample 34 we cannot find the pattern to
cut but the leftoverWidth can put other panel in
            leftoverWidth = 0;
            continue;
          }
        }
        indexOfMinHeightGapCopy.clear();
      }
      System.out.println();
      System.out.print("Total Waste for each sheet: " + currentLeftOverArea);
```

```
            System.out.print("\nPanel fit: " + numberPanelPerSheet);
            System.out.println();
            if (smallestLeftOverArea > currentLeftOverArea) {
              smallestLeftOverArea = currentLeftOverArea;
              smallestLeftOverSheet = i;
              panelAmountWithMinimumWaste = numberPanelPerSheet;
              pHeightSelected = (ArrayList)pHeightArray.clone();
              pWidthSelected = (ArrayList)pWidthArray.clone();
              xOrdinateSelected = (ArrayList)xOrdinateArray.clone();
              yOrdinateSelected = (ArrayList)yOrdinateArray.clone();
              chosenPanelIndex = (ArrayList)usedPanelIndex.clone();
              chosenPanelAmount = (ArrayList)usedPanelAmount.clone();
            }
            for(int ind =0; ind < chosenPanelIndex.size(); ind++) {
              System.out.println("Used Index " + chosenPanelIndex.get(ind));
              System.out.println("Used Amount " + chosenPanelAmount.get(ind));
            }
            pHeightArray.clear();
            pWidthArray.clear();

            xOrdinateArray.clear();
            yOrdinateArray.clear();

            usedPanelIndex.clear();
            usedPanelAmount.clear();
          }
          System.out.println();
          System.out.println(panelAmountWithMinimumWaste + " panels are cut from
panel set by putting in sheet number " + (smallestLeftOverSheet+1) + " (" +
sWidth.get(smallestLeftOverSheet) + " x " + sHeight.get(smallestLeftOverSheet) + ")
with the minimum waste " + smallestLeftOverArea);
          double minimumSheet =
10000000;//Math.ceil(nPanel.get(chosenPanelIndex.get(0)) /
chosenPanelAmount.get(0))
          double result;
          for(int d=0; d < chosenPanelIndex.size(); d++) {
            result = Math.ceil(nPanel.get(chosenPanelIndex.get(d)) /
chosenPanelAmount.get(d));
            if (minimumSheet > result) {
              minimumSheet = result;
            }
            System.out.println("To fulfill demand of " +
nPanel.get(chosenPanelIndex.get(d)) + " we need to use " + result + " sheets");
          }
          System.out.println("\nTherefore: To satisfy the demand we need: " +
minimumSheet + " sheets.");
          //To find the over cut panel
```

```
        double totalWasteOfOverCutPanel = 0;
        for (int l = 0; l < chosenPanelIndex.size(); l++) {
          overUnderCutPanel.add((minimumSheet * chosenPanelAmount.get(l)) -
nPanel.get(chosenPanelIndex.get(l)));
          if (overUnderCutPanel.get(l)<= 0) {
            overCutPanel.add(0.0);
          } else {
            overCutPanel.add(overUnderCutPanel.get(l));
          }
        }
        for (int l = 0; l < chosenPanelIndex.size(); l++) {
        wasteForEachOverCutPanelForEachSheet.add(overCutPanel.get(l)*
pHeight.get(chosenPanelIndex.get(l)) * pWidth.get(chosenPanelIndex.get(l)));
        totalWasteOfOverCutPanel = totalWasteOfOverCutPanel +
wasteForEachOverCutPanelForEachSheet.get(l);//To find the waste of the over cut
panel
        System.out.println("Over cut Panel = " + overCutPanel);
       }
      totalWastePerSheet = (smallestLeftOverArea * minimumSheet) +
totalWasteOfOverCutPanel;//Total waste per sheet included the over cut panel
      totalWastePerSheetArray.add(totalWastePerSheet);
      int index;
      double amount;
      double demand;
      for(int ind = 0; ind < chosenPanelIndex.size(); ind++) {
        System.out.println("Chosen Index " + chosenPanelIndex.get(ind));
        System.out.println("Used Amount " + chosenPanelAmount.get(ind));
        row = sheet.createRow(startRow + r);

        index = chosenPanelIndex.get(ind);
        amount = chosenPanelAmount.get(ind);
        demand = nPanel.get(chosenPanelIndex.get(ind));
        leftOverDemand.add(demand - (amount * minimumSheet));
        count = 0;

        if (ind == 0) {
          cell = row.createCell(numberingCell);
          cell.setCellValue(printNo+1);

          cell = row.createCell(sHeightCell);
          cell.setCellValue(sHeight.get(smallestLeftOverSheet));

          cell = row.createCell(sWidthCell);
          cell.setCellValue(sWidth.get(smallestLeftOverSheet));

          if (leftOverDemand.get(ind)> 0) {
            cell = row.createCell(leftOverPanelCell);
```

```
            cell.setCellValue(leftOverDemand.get(ind));
          }
        }
        if (ind < chosenPanelIndex.size()-1) { //do not let it increase the row when it is
the last panel that we put coz we have to put other data on the last row
          cell = row.createCell(demandCell);
          cell.setCellValue(nPanel.get(chosenPanelIndex.get(ind)));

          cell = row.createCell(overCutPanelCell);
          cell.setCellValue(overCutPanel.get(ind));

          cell = row.createCell(numberPanelPerSheetCell);
          cell.setCellValue(chosenPanelAmount.get(ind));

          if (leftOverDemand.get(ind)> 0) {
            cell = row.createCell(leftOverPanelCell);
            cell.setCellValue(leftOverDemand.get(ind));
          }
          // to spread the panel of cutting such that i can put the coordinate
          for (int l = 0; l < chosenPanelAmount.get(ind); l++) {
            cell = row.createCell(pHeightCell);
            //cell.setCellValue(pHeight.get(chosenPanelIndex.get(ind)));
            cell.setCellValue(pHeightSelected.get(l));
            cell = row.createCell(pWidthCell);

            //cell.setCellValue(pWidth.get(chosenPanelIndex.get(ind)));
            cell.setCellValue(pWidthSelected.get(l));
            cell = row.createCell(xCell);

            cell.setCellValue(xOrdinateSelected.get(l));
            cell = row.createCell(yCell);

            cell.setCellValue(yOrdinateSelected.get(l));
            count = count + 1;
            r++;
            row = sheet.createRow(startRow + r);
          }
          //Remove that panel use such that it can print of other panel
          for (int b = (count-1) ; b >= 0; b--) {
            pHeightSelected.remove(b);
            pWidthSelected.remove(b);

            xOrdinateSelected.remove(b);
            yOrdinateSelected.remove(b);
          }
        } else if ((chosenPanelIndex.size()-1) == ind) {
          cell = row.createCell(demandCell);
```

```
    cell.setCellValue(nPanel.get(chosenPanelIndex.get(ind)));

    cell = row.createCell(overCutPanelCell);
    cell.setCellValue(overCutPanel.get(ind));

    cell = row.createCell(numberPanelPerSheetCell);
    cell.setCellValue(chosenPanelAmount.get(ind));

    if (leftOverDemand.get(ind)> 0) {
      cell = row.createCell(leftOverPanelCell);
      cell.setCellValue(leftOverDemand.get(ind));
    }
    for (int l = 0; l < chosenPanelAmount.get(ind); l++) {
      cell = row.createCell(pHeightCell);
      //cell.setCellValue(pHeight.get(chosenPanelIndex.get(ind)));

      cell.setCellValue(pHeightSelected.get(l));
      cell = row.createCell(pWidthCell);
      //cell.setCellValue(pWidth.get(chosenPanelIndex.get(ind)));
      cell.setCellValue(pWidthSelected.get(l));
      cell = row.createCell(xCell);

      cell.setCellValue(xOrdinateSelected.get(l));
      cell = row.createCell(yCell);

      cell.setCellValue(yOrdinateSelected.get(l));
      count = count + 1;
      r++;
      row = sheet.createRow(startRow + r);
    }
    for (int b = (count-1) ; b >= 0; b--) {
      pHeightSelected.remove(b);
      pWidthSelected.remove(b);

      xOrdinateSelected.remove(b);
      yOrdinateSelected.remove(b);
    }
    r--;
  }
}
wasteForEachOverCutPanelForEachSheet.clear();
overUnderCutPanel.clear();
overCutPanel.clear();
r = r + 2;
printNo++;

cell = row.createCell(wastePerSheetCell);
```

```
        cell.setCellValue(smallestLeftOverArea);

        cell = row.createCell(numberSheetNeededToCut);
        cell.setCellValue(minimumSheet);

        cell = row.createCell(totalWastePerSheetCell);
        cell.setCellValue(totalWastePerSheet);

        endTime = System.currentTimeMillis();
        System.out.println("Time taken for this process are: " + (endTime - startTime) +
" milli seconds" );
        cell = row.createCell(computeTimeCell);
        cell.setCellValue(endTime - startTime);

        for( int d = chosenPanelIndex.size()-1; d >= 0; d--) {
          index = chosenPanelIndex.indexOf(Collections.max(chosenPanelIndex));
          amount = chosenPanelAmount.get(index);
          demand = nPanel.get(chosenPanelIndex.get(index));
          int newIndex = chosenPanelIndex.get(index);
          if (leftOverDemand.get(index) > 0) {
            nPanel.set(newIndex, leftOverDemand.get(index));  //Here we have to set it in
a different way
            chosenPanelIndex.remove(index);
            chosenPanelAmount.remove(index);
            leftOverDemand.remove(index);
          } else {
            nPanel.remove(newIndex);
            pHeight.remove(newIndex);
            pWidth.remove(newIndex);
            chosenPanelIndex.remove(index);
            chosenPanelAmount.remove(index);
            leftOverDemand.remove(index);
          }
        }
        leftOverDemand.clear();
        System.out.println("----------------------------------------------------------------------
--------------");
        j--;
        double wasteForCutting = 0;
        for (int l = 0; l < totalWastePerSheetArray.size(); l++) {
          wasteForCutting = wasteForCutting + totalWastePerSheetArray.get(l);
        }
        cell = row.createCell(wasteForCuttingCell);
        cell.setCellValue(wasteForCutting);
      }
      chosenPanelIndex.clear();
      chosenPanelAmount.clear();
```

```
  file.close();

  FileOutputStream outFile =new FileOutputStream(new File(excelFile));
  workbook.write(outFile);
  outFile.close();
  } catch (FileNotFoundException e) {
    e.printStackTrace();
  } catch (IOException e) {
    e.printStackTrace();
  }
}
public static void ReadExcel(ArrayList<Double> sWidth,
                             ArrayList<Double> sHeight, ArrayList<Double> nSheet,
                             ArrayList<Double> pWidth, ArrayList<Double> pHeight,
                             ArrayList<Double> nPanel, String excelFile) {
  try {
    FileInputStream file = new FileInputStream(new File(excelFile));

    //Get the workbook instance for XLSX file
    XSSFWorkbook workbook = new XSSFWorkbook(file);

    //Get first sheet from the workbook
    XSSFSheet sheet = workbook.getSheet("Input");// or .getsheetAt(2);

    // cRow = currentRow, pRow = processingRow
    int cRow = 0 , pRow = 0;          // cRow can give any values but pRow cannot
    int cColumn = 1;          // here we also can change it
    dataInRow = 0;

    //Iterate through each rows from first sheet
    Iterator<Row> rowIterator = sheet.iterator();
    while(rowIterator.hasNext()) {
      Row row = rowIterator.next();
      cRow++;
      cColumn = 1;
      dataInRow = 0;
      //For each row, iterate through each columns
      Iterator<Cell> cellIterator = row.cellIterator();
      while(cellIterator.hasNext()) {
        Cell cell = cellIterator.next();
        switch(cell.getCellType()) {
        case Cell.CELL_TYPE_BOOLEAN:
          System.out.print(cell.getBooleanCellValue() + "\t\t");
          dataInRow ++;
          break;
        case Cell.CELL_TYPE_NUMERIC:
          System.out.print(cell.getNumericCellValue() + "\t\t");
```

```
            if(cRow>pRow) {
              pRow = cRow;
            } else {
              switch(cColumn) {
              case 1:
                pHeight.add(cell.getNumericCellValue());
                break;
              case 2:
                pWidth.add(cell.getNumericCellValue());
                break;
              case 3:
                nPanel.add(cell.getNumericCellValue());
                break;
              case 4:
                sHeight.add(cell.getNumericCellValue());
                break;
              case 5:
                sWidth.add(cell.getNumericCellValue());
                break;
              case 6:
                nSheet.add(cell.getNumericCellValue());
              }
              cColumn++;
            }
            dataInRow++;
            break;
          case Cell.CELL_TYPE_STRING:
            System.out.print(cell.getStringCellValue() + "\t\t");
            dataInRow++;
            break;
          }
        }
        if(dataInRow > 0) {
          System.out.println("");
        }
      }
      file.close();
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

7.2. Appendix B: Input Twenty Testing Instances

These are all the twenty different size of instances that are used to run in all techniques. These instances have to sort vertical or horizontally for the width or height in accordance with the regulation for each technique before running it. Again, the total number of sheet in stock is unlimited for all techniques. Later, other techniques by considering the number of sheet in stock is developed.

Table 7.1 Instance 1

No	pHeight	pWidth	nPanel	sHeight	sWidth	nSheet
1	25.55	20.70	107.00	51.00	42.00	10000
2	25.50	20.90	112.00	50.50	42.00	10000

Table 7.2 Instance 2

No	pHeight	pWidth	nPanel	sHeight	sWidth	nSheet
1	20.15	24.00	67.00	40.00	48.00	10000
2	20.15	24.00	234.00	36.00	48.00	10000

Table 7.3 Instance 3

No	pHeight	pWidth	nPanel	sHeight	sWidth	nSheet
1	20.50	21.50	375.00	45.00	42.00	10000
2	15.00	21.25	200.00	40.50	43.00	10000

Table 7.4 Instance 4

No	pHeight	pWidth	nPanel	sHeight	sWidth	nSheet
1	26.8	20.15	244.00	58.50	48.00	10000
2	25.45	19.45	93.00	53.50	42.00	10000
3	19.45	24	200.00	50.50	42.00	10000

Table 7.5 Instance 5

No	pHeight	pWidth	nPanel	sHeight	sWidth	nSheet
1	22.25	22.90	116.00	40.00	48.00	10000
2	21.70	22.90	141.00	26.80	48.00	10000
3	19.60	22.90	295.00	24.00	48.00	10000

Table 7.6 Instance 6

No	pHeight	pWidth	nPanel	sHeight	sWidth	nSheet
1	25.00	20.00	150.00	50.50	48.00	10000
2	25.00	20.00	150.00	49.50	42.00	10000
3	9999	9999	0.00	42.00	48.00	10000
4	9999	9999	0.00	41.00	43.00	10000

Table 7.7 Instance 7

No	pHeight	pWidth	nPanel	sHeight	sWidth	nSheet
1	14.00	25.90	209.00	51.50	42.00	10000
2	9999	9999	0.00	44.00	49.00	10000
3	9999	9999	0.00	43.50	48.00	10000
4	9999	9999	0.00	40.50	48.00	10000
5	9999	9999	0.00	37.00	49.00	10000

Table 7.8 Instance 8

No	pHeight	pWidth	nPanel	sHeight	sWidth	nSheet
1	25.95	24.20	200.00	51.50	48.00	10000
2	25.95	24.20	239.00	50.50	49.00	10000
3	25.95	24.20	180.00	47.50	49.00	10000
4	25.95	24.20	350.00	47.00	49.00	10000
5	23.60	24.50	223.00	43.50	48.00	10000
6	16.85	24.50	300.00	0.00	0.00	0.00

Table 7.9 Instance 9

No	pHeight	pWidth	nPanel	sHeight	sWidth	nSheet
1	28.00	22.50	426.00	42.00	48.00	10000
2	26.50	22.50	75.00	40.00	48.00	10000
3	26.50	22.50	500.00	28.00	48.00	10000
4	26.50	22.50	659.00	26.80	48.00	10000
5	26.00	22.50	60.00	24.00	48.00	10000
6	21.00	23.00	292.00	0.00	0.00	0.00
7	20.00	22.50	99.00	0.00	0.00	0.00
8	19.50	23.00	129.00	0.00	0.00	0.00

Table 7.10 Instance 10

No	pHeight	pWidth	nPanel	sHeight	sWidth	nSheet
1	26.00	24.00	912.00	51.50	48.00	10000
2	20.50	23.00	417.00	48.50	48.00	10000
3	9999	9999	0.00	46.50	48.00	10000

No	pHeight	pWidth	nPanel	sHeight	sWidth	nSheet
4	9999	9999	0.00	43.50	49.00	10000
5	9999	9999	0.00	42.50	48.00	10000
6	9999	9999	0.00	41.50	48.00	10000

Table 7.11 Instance 11

No	pHeight	pWidth	nPanel	sHeight	sWidth	nSheet
1	20.70	23.50	80.00	53.50	48.00	10000
2	20.50	22.20	21.00	52.50	48.00	10000
3	19.00	23.00	118.00	49.50	48.00	10000
4	9999	9999	0.00	42.00	48.00	10000
5	9999	9999	0.00	40.00	48.00	10000
6	9999	9999	0.00	36.00	48.00	10000

Table 7.12 Instance 12

No	pHeight	pWidth	nPanel	sHeight	sWidth	nSheet
1	26.00	22.50	59.00	53.00	48.00	10000
2	22.50	24.00	60.00	52.50	48.00	10000
3	22.50	24.00	60.00	51.50	48.00	10000
4	22.50	24.00	60.00	51.00	48.00	10000
5	22.50	24.00	119.00	44.50	48.00	10000
6	22.50	24.00	119.00	42.00	48.00	10000
7	22.50	24.00	119.00	40.00	48.00	10000
8	9999	9999	0.00	36.00	48.00	10000

Table 7.13 Instance 13

No	pHeight	pWidth	nPanel	sHeight	sWidth	nSheet
1	26.20	22.70	858	58.00	48.00	10000
2	24.00	24.25	375	55.50	42.00	10000
3	21.85	22.65	959	55.00	48.00	10000
4	9999	9999	0.00	54.50	42.00	10000
5	9999	9999	0.00	52.00	48.00	10000
6	9999	9999	0.00	50.50	42.00	10000
7	9999	9999	0.00	49.50	48.00	10000
8	9999	9999	0.00	47.50	48.00	10000
9	9999	9999	0.00	43.50	48.00	10000

Table 7.14 Instance 14

No	pHeight	pWidth	nPanel	sHeight	sWidth	nSheet
1	22.85	23.20	91.00	52.00	48.00	10000
2	20.75	26.50	342.00	50.00	48.00	10000
3	20.75	26.50	570.00	42.00	48.00	10000
4	20.75	26.50	1020.00	40.00	48.00	10000
5	20.65	27.45	99.00	36.00	48.00	10000
6	17.45	22.90	154.00	34.00	48.00	10000
7	17.00	22.25	175.00	28.00	48.00	10000
8	16.25	22.00	283.00	26.80	48.00	10000
9	16.00	22.25	334.00	24.00	48.00	10000

Table 7.15 Instance 15

No	pHeight	pWidth	nPanel	sHeight	sWidth	nSheet
1	27.00	23.50	86.00	55.50	48.00	10000
2	25.00	20.50	53.00	50.50	42.00	10000
3	21.00	24.50	160.00	48.00	49.00	10000
4	16.00	24.50	4000.00	43.50	43.00	10000
5	9999	9999	0.00	41.50	49.00	10000
6	9999	9999	0.00	40.50	48.00	10000
7	9999	9999	0.00	39.50	49.00	10000
8	9999	9999	0.00	37.00	43.00	10000
9	9999	9999	0.00	28.00	48.00	10000
10	9999	9999	0.00	26.80	48.00	10000
11	9999	9999	0.00	24.00	48.00	10000

Table 7.16 Instance 16

No	pHeight	pWidth	nPanel	sHeight	sWidth	nSheet
1	26.50	22.00	160.00	57.00	48.00	10000
2	26.50	22.00	204.00	55.50	49.00	10000
3	21.00	24.50	60.00	53.50	48.00	10000
4	21.00	24.50	70.00	52.50	43.00	10000
5	20.00	23.50	254.00	52.50	48.00	10000
6	9999	9999	0.00	50.50	42.00	10000
7	9999	9999	0.00	49.50	42.00	10000
8	9999	9999	0.00	41.50	48.00	10000
9	9999	9999	0.00	41.50	49.00	10000
10	9999	9999	0.00	40.50	49.00	10000
11	9999	9999	0.00	39.50	48.00	10000

Table 7.17 Instance 17

No	pHeight	pWidth	nPanel	sHeight	sWidth	nSheet
1	21.00	24.40	200.00	59.50	48.00	10000
2	20.70	24.50	123.00	54.00	48.00	10000
3	20.70	24.50	112.00	53.50	48.00	10000
4	20.55	24.10	134.00	52.00	48.00	10000
5	20.55	24.00	167.00	49.00	48.00	10000
6	20.50	24.00	123.00	45.00	48.00	10000
7	20.50	24.05	445.00	42.50	48.00	10000
8	20.45	24.05	167.00	41.50	49.00	10000
9	20.45	24.00	223.00	41.00	48.00	10000
10	20.00	21.00	129.00	41.00	49.00	10000
11	19.70	24.30	2556.00	40.50	48.00	10000
12	17.30	23.05	50.00	40.50	49.00	10000
13	9999	9999	0.00	40.00	48.00	10000
14	9999	9999	0.00	40.00	48.00	10000
15	9999	9999	0.00	39.50	42.00	10000

Table 7.18 Instance 18

No	pHeight	pWidth	nPanel	sHeight	sWidth	nSheet
1	27.50	22.90	319.00	58.50	49.00	10000
2	27.10	20.80	90.00	55.50	42.00	10000
3	21.00	24.00	211.00	55.00	42.00	10000
4	20.50	24.00	286.00	54.50	42.00	10000
5	20.45	24.00	150.00	54.50	48.00	10000
6	16.75	21.50	65.00	54.00	42.00	10000
7	9999	9999	0.00	53.50	48.00	10000
8	9999	9999	0.00	52.50	42.00	10000
9	9999	9999	0.00	52.50	48.00	10000
10	9999	9999	0.00	51.50	48.00	10000
11	9999	9999	0.00	51.00	42.00	10000
12	9999	9999	0.00	50.50	43.00	10000
13	9999	9999	0.00	43.00	48.00	10000
14	9999	9999	0.00	41.50	48.00	10000
15	9999	9999	0.00	41.00	48.00	10000
16	9999	9999	0.00	40.50	48.00	10000

Table 7.19 Instance 19

No	pHeight	pWidth	nPanel	sHeight	sWidth	nSheet
1	27.9	23.5	225.00	58.50	48.00	10000
2	27.35	20.45	240.00	55.50	42.00	10000
3	27.3	22.3	2,477.00	55.50	48.00	10000
4	26.4	22.2	109.00	55.00	42.00	10000
5	26.3	22.3	780.00	54.50	42.00	10000
6	26.25	22.5	1,180.00	51.50	42.00	10000
7	25.65	22.2	3.00	51.00	48.00	10000
8	25.65	22.2	90.00	49.00	42.00	10000
9	25.5	22.2	84.00	49.00	48.00	10000
10	25.5	22.2	199.00	42.50	48.00	10000
11	25.5	22.2	84.00	42.00	48.00	10000
12	24.1	16	97.00	41.00	48.00	10000
13	24	23.35	71.00	40.00	48.00	10000
14	23.85	24.15	70.00	36.00	48.00	10000
15	21.5	15.75	556.00	28.00	48.00	10000
16	20.5	24	98.00	26.80	48.00	10000
17	20.3	22.3	850.00	24.00	48.00	10000
18	20.3	22.3	1,159.00	0.00	0.00	0.00
19	20.1	23.4	40.00	0.00	0.00	0.00
20	19.9	23.4	534.00	0.00	0.00	0.00
21	19.5	23.75	108.00	0.00	0.00	0.00
22	19.5	23.4	259.00	0.00	0.00	0.00
23	19.15	23.4	157.00	0.00	0.00	0.00
24	18.5	24	94.00	0.00	0.00	0.00
25	16.6	21.15	840.00	0.00	0.00	0.00
26	15.75	22.25	96.00	0.00	0.00	0.00
27	15.75	22.25	111.00	0.00	0.00	0.00

Table 7.20 Instance 20

No	pHeight	pWidth	nPanel	sHeight	sWidth	nSheet
1	25.85	24.20	500.00	56.50	49.00	10000
2	25.85	24.20	970.00	55.50	48.00	10000
3	25.50	14.00	75.00	54.00	48.00	10000
4	25.40	22.25	226.00	53.50	48.00	10000
5	22.75	16.00	267.00	53.00	42.00	10000
6	22.75	16.25	500.00	53.00	48.00	10000
7	22.25	23.40	135.00	52.50	48.00	10000
8	22.25	16.30	182.00	52.00	42.00	10000
9	22.20	23.30	130.00	51.50	48.00	10000
10	21.50	23.40	120.00	51.00	48.00	10000
11	21.30	23.40	112.00	50.50	42.00	10000
12	21.00	24.25	134.00	50.50	48.00	10000
13	18.50	24.00	131.00	49.50	48.00	10000
14	17.00	22.75	129.00	47.00	48.00	10000
15	16.80	23.00	179.00	45.00	48.00	10000
16	9999	9999	0.00	45.00	49.00	10000
17	9999	9999	0.00	44.50	48.00	10000
18	9999	9999	0.00	44.00	48.00	10000
19	9999	9999	0.00	44.00	49.00	10000
20	9999	9999	0.00	42.50	48.00	10000
21	9999	9999	0.00	42.50	49.00	10000
22	9999	9999	0.00	41.50	48.00	10000
23	9999	9999	0.00	41.00	48.00	10000
24	9999	9999	0.00	40.50	43.00	10000
25	9999	9999	0.00	40.50	48.00	10000
26	9999	9999	0.00	40.00	48.00	10000

7.3. Appendix C: Output Twenty Testing Instances

Since there are so many instances, only the first instance and its output pattern are given for each technique of the heuristic methods.

7.3.1. 2D Simple Heuristic Cutting

Table 7.21 The output for the first instance using 2DSHC

NO	Sheet		Panel			Minimum	Number of	Number of sheet	Leftover	Compute	Coordinate		Over Cut	waste/sheet -	Total Waste
	Height	Width	height	width	Demand	Waste/sheet	panel/sheet	that we will cut	panel	time	x	y	Panel	over cut panel	For Cutting
1	50.5	42	25.55	20.7	107		2				0	0	1		
			25.55	20.7							20.7	0			
						1063.23		54		1258				57943.31	57943.31
2	51	42	25.5	20.9	112		4				0	0	0		
			25.5	20.9							20.9	0			
			25.5	20.9							0	25.5			
			25.5	20.9							20.9	25.5			
						10.2		28		1263				285.6	58228.91

7.3.2. 2D Horizontal Construction

Table 7.22 The output for the first instance using 2DHC

Sheet		Panel		Level	Waste	Time	Total Waste
Height	Width	height	width				
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1305	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1307	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1309	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1310	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1312	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1314	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1315	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1317	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1318	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1320	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1322	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1324	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1325	
51	42	25.55	20.7	1			

Sheet		Panel		Level	Waste	Time	Total Waste
Height	Width	height	width				
		25.55	20.7	1	1084.23	1327	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1330	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1332	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1333	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1335	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1337	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1339	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1341	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1342	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1344	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1345	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1347	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1349	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1350	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1351	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1352	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1353	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1354	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1355	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1356	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1357	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1358	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1359	

Sheet		Panel		Level	Waste	Time	Total Waste
Height	Width	height	width				
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1360	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1361	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1362	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1363	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1364	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1365	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1366	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1368	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1369	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1370	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1371	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1372	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1372	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1373	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1374	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1375	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1375	
51	42	25.55	20.7	1			
		25.5	20.9	1	1080.165	1376	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1377	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1378	
51	42	25.5	20.9	1			

Sheet		Panel		Level	Waste	Time	Total Waste
Height	Width	height	width				
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1380	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1381	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1382	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1383	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1384	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1385	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1386	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1387	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1388	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1394	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1395	
51	42	25.5	20.9	1			
		25.5	20.9	1			

Sheet		Panel		Level	Waste	Time	Total Waste
Height	Width	height	width				
		25.5	20.9	2			
		25.5	20.9	2	10.2	1397	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1398	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1400	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1401	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1403	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1404	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1405	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1406	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1407	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1408	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1409	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			

Sheet		Panel		Level	Waste	Time	Total Waste
Height	Width	height	width				
		25.5	20.9	2	10.2	1410	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1411	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1411	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2	543.15	1411	59362.91

7.3.3. 2D Vertical Construction

Table 7.23 The output for the first instance using 2DVC

Sheet		Panel		Level	Waste	Time	Total Waste
Height	Width	height	width				
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1328	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1332	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1335	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1339	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1342	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1345	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			

Sheet		Panel		Level	Waste	Time	Total Waste
Height	Width	height	width				
		25.5	20.9	2	10.2	1348	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1351	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1353	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1356	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1359	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1362	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1365	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1368	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1370	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1372	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1374	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1376	

Sheet		Panel		Level	Waste	Time	Total Waste
Height	Width	height	width				
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1379	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1382	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1384	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1386	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1388	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1390	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1392	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1394	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1396	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1397	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1398	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1399	
51	42	25.55	20.7	1			

Sheet		Panel		Level	Waste	Time	Total Waste
Height	Width	height	width				
		25.55	20.7	2	1084.23	1400	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1401	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1401	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1402	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1403	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1404	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1405	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1406	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1407	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1408	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1409	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1409	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1410	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1411	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1411	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1412	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1413	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1414	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1415	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1415	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1416	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1417	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1418	

Sheet		Panel		Level	Waste	Time	Total Waste
Height	Width	height	width				
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1418	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1419	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1420	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1420	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1421	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1422	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1422	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1423	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1424	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1424	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1425	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1426	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1426	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1427	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1427	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1428	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1429	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1429	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1430	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1430	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1431	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1431	
51	42	25.55	20.7	1			

Sheet		Panel		Level	Waste	Time	Total Waste
Height	Width	height	width				
		25.55	20.7	2	1084.23	1432	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1432	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1433	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1433	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1434	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1434	
51	42	25.55	20.7	1	1613.115	1434	59362.91

7.3.4. 2D Horizontal Improvement

Table 7.24 The output for the first instance using 2DHI

NO	Sheet		Panel			Minimum	Number of	Number of sheet	Leftover	Compute	Coordinate		Over Cut	waste/sheet -	Total Waste
	Height	Width	height	width	Demand	Waste/sheet	panel/sheet	that we will cut	panel	time	x	y	Panel	over cut panel	For Cutting
1	50.5	42	25.55	20.7	107		2				0	0	1		
			25.55	20.7							20.7	0			
						1063.23		54		1161				57943.31	57943.31
2	51	42	25.5	20.9	112		4				0	0	0		
			25.5	20.9							20.9	0			
			25.5	20.9							0	25.5			
			25.5	20.9							20.9	25.5			
						10.2		28		1168				285.6	58228.91

7.3.5. 2D Vertical Improvement

Table 7.25 The output for the first instance using 2DVI

NO	Sheet		Panel			Minimum	Number of	Number of sheet	Leftover	Compute	Coordinate		Over Cut	waste/sheet -	Total Waste
	Height	Width	height	width	Demand	Waste/sheet	panel/sheet	that we will cut	panel	time	x	y	Panel	over cut panel	For Cutting
1	51	42	25.5	20.9	112		4				0	0	0		
			25.5	20.9							0	25.5			
			25.5	20.9							20.9	0			
			25.5	20.9							20.9	25.5			
						10.2		28		1314				285.6	285.6
2	50.5	42	25.55	20.7	107		2				0	0	1		
			25.55	20.7							20.7	0			
						1063.23		54		1319				57943.31	58228.91

7.3.6. Sheet Width Panel Height Horizontal Cut

Table 7.26 The output of the first instance using SW_PH_HC

Sheet		Panel		Level	Waste	Time	Total Waste
Height	Width	height	width				
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1152	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1155	
51	42	25.55	20.7	1			

Sheet		Panel		Level	Waste	Time	Total Waste
Height	Width	height	width				
		25.55	20.7	1	1084.23	1157	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1159	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1161	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1164	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1166	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1169	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1171	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1174	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1175	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1177	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1179	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1180	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1182	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1184	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1186	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1188	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1189	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1191	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1193	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1195	

Sheet		Panel		Level	Waste	Time	Total Waste
Height	Width	height	width				
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1201	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1202	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1204	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1206	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1207	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1209	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1210	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1211	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1213	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1214	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1215	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1217	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1218	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1219	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1220	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1221	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1222	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1224	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1225	
51	42	25.55	20.7	1			

Sheet		Panel		Level	Waste	Time	Total Waste
Height	Width	height	width				
		25.55	20.7	1	1084.23	1226	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1228	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1229	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1230	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1231	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1232	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1233	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1234	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1235	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1236	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1237	
51	42	25.55	20.7	1			
		25.55	20.7	1	1084.23	1238	
51	42	25.55	20.7	1			
		25.5	20.9	1	1080.17	1239	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1240	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1241	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1243	
51	42	25.5	20.9	1			
		25.5	20.9	1			

Sheet		Panel		Level	Waste	Time	Total Waste
Height	Width	height	width				
		25.5	20.9	2			
		25.5	20.9	2	10.2	1245	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1246	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1248	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1249	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1250	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1251	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1252	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1254	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1255	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1258	
51	42	25.5	20.9	1			

Sheet		Panel		Level	Waste	Time	Total Waste
Height	Width	height	width				
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1259	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1260	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1261	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1262	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1263	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1264	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1265	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1266	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1267	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1268	

Sheet		Panel		Level	Waste	Time	Total Waste
Height	Width	height	width				
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1269	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1270	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1271	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1272	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2	543.15	1272	59362.9

7.3.7. Sheet Width Panel Width Vertical Cut

Table 7.27 The output of the first instance using SW_PW_VC

Sheet		Panel		Level	Waste	Time	Total Waste
Height	Width	height	width				
	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1057	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1064	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1069	
51	42	25.5	20.9	1			
		25.5	20.9	1			

Sheet		Panel		Level	Waste	Time	Total Waste
Height	Width	height	width				
		25.5	20.9	2			
		25.5	20.9	2	10.2	1075	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1080	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1084	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1088	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1094	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1099	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1102	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1105	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1108	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1111	
51	42	25.5	20.9	1			

Sheet		Panel		Level	Waste	Time	Total Waste
Height	Width	height	width				
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1114	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1116	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1119	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1121	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1124	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1127	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1129	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1131	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1133	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1135	

Sheet		Panel		Level	Waste	Time	Total Waste
Height	Width	height	width				
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1136	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1138	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1140	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1142	
51	42	25.5	20.9	1			
		25.5	20.9	1			
		25.5	20.9	2			
		25.5	20.9	2	10.2	1144	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1145	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1146	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1147	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1148	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1149	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1150	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1150	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1151	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1152	
51	42	25.55	20.7	1			

Sheet		Panel		Level	Waste	Time	Total Waste
Height	Width	height	width				
		25.55	20.7	2	1084.23	1153	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1154	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1154	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1155	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1156	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1157	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1157	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1158	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1159	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1159	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1161	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1161	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1162	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1163	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1163	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1164	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1165	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1165	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1166	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1166	

Sheet		Panel		Level	Waste	Time	Total Waste
Height	Width	height	width				
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1167	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1167	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1168	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1169	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1169	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1170	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1170	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1171	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1171	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1172	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1173	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1173	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1174	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1174	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1175	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1175	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1176	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1176	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1177	
51	42	25.55	20.7	1			

Sheet		Panel		Level	Waste	Time	Total
Height	Width	height	width				Waste
		25.55	20.7	2	1084.23	1177	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1178	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1179	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1179	
51	42	25.55	20.7	1			
		25.55	20.7	2	1084.23	1180	
51	42	25.55	20.7	1	1613.115	1180	59362.905

7.3.8. Minimum Sheet Width Ordering Panel Height Horizontal Cut

Table 7.28 The output of the first instance using MinSW_OPH_HC

NO	Sheet		Panel			Minimum	Number of	Number of sheet	Leftover	Compute	Coordinate		Over Cut	waste/sheet -	Total Waste
	Height	Width	height	width	Demand	Waste/sheet	panel/sheet	that we will cut	panel	time	x	y	Panel	over cut panel	For Cutting
1	50.5	42	25.6	20.7	107		1				0	0	0		
			25.5	20.9	112		1		5		20.7	0	0		
						1059.165		107		2169				113330.655	113330.66
2	51	42	25.5	20.9	5		4				0	0	3		
			25.5	20.9							20.9	0			
			25.5	20.9							0	25.5			
			25.5	20.9							20.9	25.5			
						10.2		2		2177				1619.25	114949.91

7.3.9. Minimum Sheet Height Ordering Panel Width Vertical Cut

Table 7.29 The output of the first instance using MinSH_OPW_VC

NO	Sheet		Panel			Minimum	Number of	Number of sheet	Leftover	Compute	Coordinate		Over Cut	waste/sheet -	Total Waste
	Height	Width	height	width	Demand	Waste/sheet	panel/sheet	that we will cut	panel	time	x	y	Panel	over cut panel	For Cutting
1	51	42	25.5	20.9	112		4				0	0	0		
			25.5	20.9							0	25.5			
			25.5	20.9							20.9	0			
			25.5	20.9							20.9	25.5			
						10.2		28		1063				285.6	285.6
2	50.5	42	25.6	20.7	107		2				0	0	1		
			25.6	20.7							20.7	0			
						1063.23		54		1069				57943.305	58228.905