VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

MATHEMATICAL MODELING (CO2011)

Assignment

"Cutting stock problem"

Instructor(s): MSc. Trần Hồng Tài

Students: Nguyễn Đăng Duy - 2252116 *(Group CC02 - Team 08)*
Bùi Thái An - 2252001 *(Group CC02 - Team 08, **Leader**)*
Phạm Nguyễn Hải Khánh - 2252333 *(Group CC02 - Team 08)*
Lê Thị Thúy Hằng - 2052460 *(Group CC04 - Team 08)*

HO CHI MINH CITY, DECEMBER 2024

Contents

List of Symbols

$\mathbb{N}$ Set of natural numbers

$\mathbb{R}$ Set of real numbers

$\mathbb{R}^+$ Set of positive real numbers

List of Acronyms

ODE (First-Order) Ordinary Differential Equation

IVP Initial-Value Problem

LTE Local Truncation Error

DS Dynamical System

Fig. Figure

Tab. Table

Sys. System of Equations

Eq. Equation

e.g. For Example

i.e. That Is

List of Figures

List of Tables

Member list & Workload

No.	Fullname	Student ID	Tasks	% done
1	Nguyễn Đăng Duy	2252116	Introduction, Problem Modelling - Problem Modelling using Linear Programming, Evaluation	100%
2	Bùi Thái An	2213686	Finding the algorithm (Column Generation), Implementing the algorithm (Greedy)	100%
3	Phạm Nguyễn Hải Khánh	2252333	Problem Modelling - Introduce the problem, Problem-Solving Algorithm and Heuristic Approach	100%
4	Lê Thị Thúy Hằng	2052460	Finding Algorithm, Implementing Algorithm (GA, Bottom-left)	100%

Table 1: Member list & workload

Github link to the repository

The GitHub repository can be downloaded here.

Acknowledgement

I would like to express my sincere gratitude to my lecturer, Mr. Tran Hong Tai, for his invaluable contribution to this project. His insightful guidance, expert knowledge, and constructive feedback were instrumental in shaping and refining this work. I am particularly grateful for their patience, encouragement, and dedication to fostering a stimulating learning environment. Their mentorship has been invaluable, and I am deeply appreciative of the time and effort they invested in my academic development.

1 Cutting Stock Problem

1.1 Introduction

The two-dimensional rectangular guillotine cutting stock problem (2DCSP) is a crucial optimization problem with widespread applications in industries like manufacturing, packaging, textiles, and metalworking. The goal is to minimize material waste when cutting smaller rectangular pieces from larger stock sheets. This involves determining the optimal number of stock sheets and the arrangement of cuts on each sheet to fulfill customer orders while minimizing trim loss (unused material). The defining "guillotine" constraint mandates that each cut must fully traverse the sheet's width or length, creating two smaller rectangles. This reflects the operation of a guillotine cutter and common limitations in cutting processes.

1.2 History and One-Dimensional Cutting Stock Problem

The 2DCSP-MS builds upon the simpler one-dimensional cutting stock problem (1DCSP) and its variant with multiple stock sizes. Gilmore and Gomory's (1965) pioneering work on column generation for the 1DCSP laid the groundwork for tackling cutting stock problems using integer programming. In these models, columns represent feasible cutting patterns, and algorithms iteratively generate new patterns to reduce required stock. This approach can be adapted to handle multiple stock sizes in 1D. Later research, such as Suliman's (2001) work on branch and bound column generation for the 1DCSP, further advanced solution methods. These foundational developments in the 1D context inform approaches to the more challenging 2DCSP-MS.

1.3 Two-Dimensional Cutting Stock Problem

The 2DCSP-MS introduces an additional layer of complexity compared to the single-stock-size 2DCSP. The selection of the optimal stock size mix adds to the combinatorial complexity of determining cutting patterns. Direct application of integer programming, while theoretically possible, quickly becomes computationally prohibitive for realistic problem instances. Therefore, heuristic algorithms are crucial for practical solutions. Existing heuristics for the single-size 2DCSP can often be adapted, though considerations for stock size selection must be integrated. These include modified dynamic programming methods (building upon Farley, 1990b), constructive heuristics that consider different stock sizes during pattern construction (extending Wang, 1983), and adapted two-stage methods that incorporate stock size choices. The three-stage sequential heuristic by Suliman (2005) could also be extended to account for multiple stock sizes, potentially by incorporating a preliminary stock selection stage before width-cutting, length-wise arrangement, and pattern repetition. Research continues to focus on developing more effective heuristics and exact methods for the 2DCSP-MS due to the significant economic advantages of minimizing waste when utilizing diverse available stock materials.

2 Algorithm Introduction

2.1 Column Generation

2.1.1 Summary

Column generation is a technique for solving large linear optimization problems by generating only the variables that have the potential to improve the objective function. This is crucial for large problems with a vast number of variables. Formulating a problem using this technique simplifies the problem construction because not all possibilities need to be enumerated.

The method works as follows: the original problem is decomposed into two problems: the master problem and the subproblem.

- The master problem is the initial column-wise formulation of the problem, considering only a subset of variables.
- The subproblem is a new problem generated to identify a new variable. Its objective function is the reduced cost of the new variable with respect to the current dual variables, and the constraints require that the variable satisfy the naturally occurring constraints. The subproblem is also referred to as the pricing problem or the restricted master problem (RMP). This implies that this method is suitable for problems where the constraint set has a well-understood combinatorial structure that allows for such a decomposition.

Several techniques exist to perform this decomposition of the original problem into the master problem and subproblem(s). The theory behind this method is based on the Dantzig-Wolfe decomposition.

In summary, when the master problem is solved, we obtain dual prices for each constraint in the master problem. This information is then used in the objective function of the subproblem. The subproblem is solved. If the objective value of the subproblem is negative, a variable with a negative reduced cost has been identified. This variable is then added to the master problem, and the master problem is re-solved.

Re-solving the master problem generates a new set of dual values, and this process is repeated until no negative reduced cost can be identified. When the subproblem returns a non-negative reduced cost solution, we can conclude that the solution to the master problem is optimal.

The stages of the column generation algorithm are shown in the following flowchart at **1**:

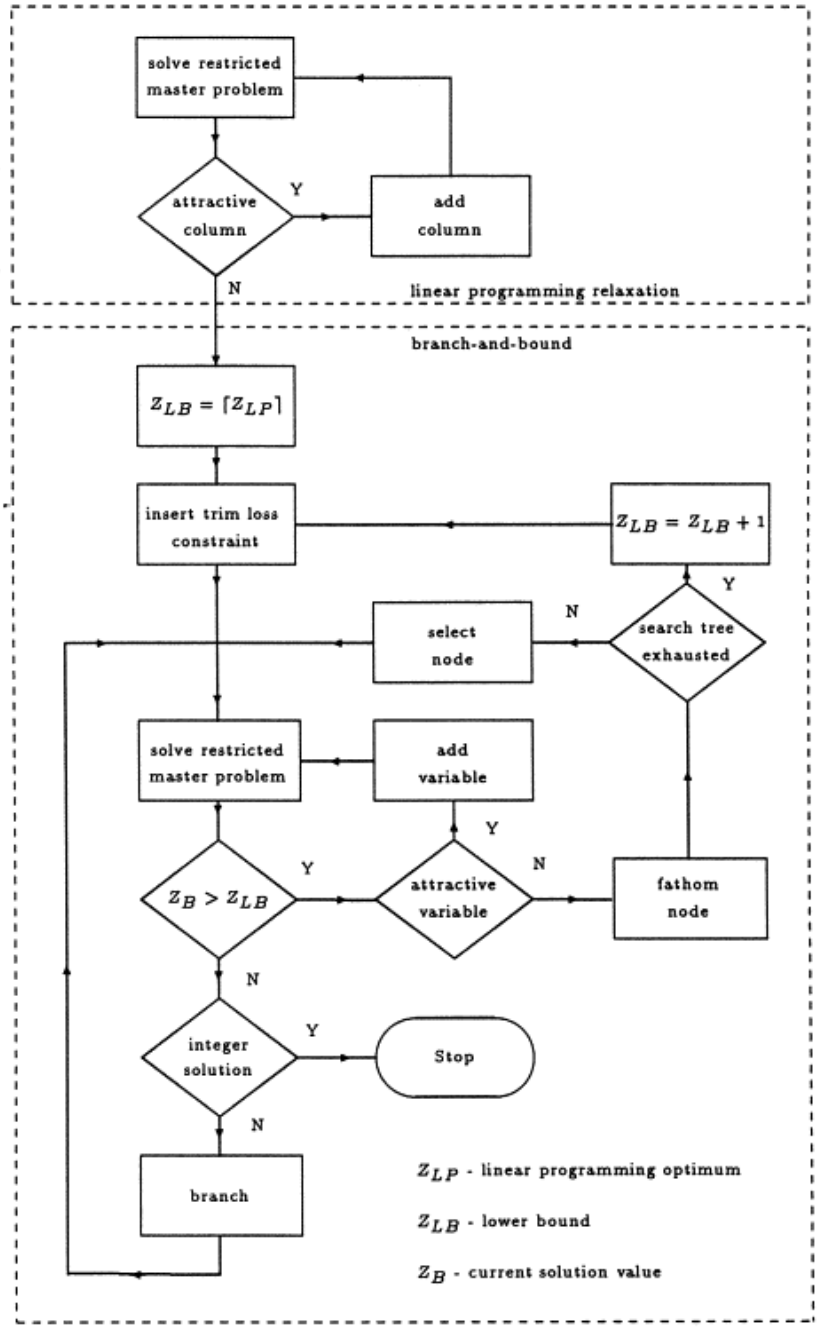

Figure 1: Column Generation Flowchart.

2.2 Genetic Algorithm

2.2.1 Summary

The Genetic Algorithm (GAs) is a cornerstone of evolutionary computation due to its wide range of applications, including the 2D cutting stock problem. GAs are inspired by the process of natural selection. They mimic the evolutionary process, where individuals with favorable traits are more likely to survive and reproduce, leading to the gradual improvement of the population over generations. [1]

The core concept of GAs involves creating a population of potential solutions, often referred to as individuals or chromosomes, then proceeds through a series of generations which typically involves the following steps:

- **Step 1 - Initialization:** A starting population of random solutions or individual is created. Each solution, or individual, represents a potential solution to the problem.
- **Step 2 - Fitness evaluation:** A fitness function is defined to assess the quality of each individual. This function assigns a fitness score to each solution based on how well it solves the problem.
- **Step 3 - Selection:** Individuals with higher fitness scores are selected to become parents for the next generation. This process can be implemented using various selection techniques, such as roulette wheel selection or tournament selection.
- **Step 4 - Crossover:** Selected parents combine their genetic material to create offspring. This mimics the biological process of reproduction, where genetic information is exchanged between parents.
- **Step 5 - Mutation:** Random changes are introduced to maintain diversity and prevent stagnation.This mimics random genetic mutations.
- **Step 6 - Replacement:** The newly created offspring replace the least fit individuals in the population.
- **Step 7 - Termination:** The process repeats until a termination criterion is met, such as reaching a maximum number of generations or finding a satisfactory solution. [2]

2.2.2 Advantages and Disadvantages in Cutting Stock Problem

Advantages

GAs have numerous advantages over traditional optimization algorithms. Two of the most notable are:

- Their ability to handle complex problems with multiple local optima. They can explore a wide range of solutions and prevent themselves from getting stuck in suboptimal regions.
- Their potential for parallelization.

GAs can tackle a wide variety of optimization problems, regardless of whether the objective (fitness) function is stationary or non-stationary (time-varying), linear or non-linear, continuous or discrete, or has stochastic noise. Because the offspring of the population behave like independent agents, the population (or any subgroup) can simultaneously explore the search space in multiple directions. This characteristic makes it highly suitable for the implementation of parallel algorithms. Different parameters, or even different encoding strings, can be manipulated simultaneously. [3]

Disadvantages

- Can be computationally expensive, especially for large-scale problems.
- The outcome of a GA can vary due to its random nature. While GAs often find good solutions, they do not guarantee optimal solutions. The performance of a GA can be sensitive to the choice of parameters, such as population size, crossover rate, and mutation rate. Any inappropriate choices can make the algorithm difficult to converge and produce meaningless results.
- In some cases, the GA may converge to a suboptimal solution prematurely before exploring the entire search space.

3 Modeling the Problem

3.1 Problem Introduction

A company X Paper mill X manufactures and distributes a type of paper with dimensions defined by standards. Suppose we have an order with requirements summarized in the following table:

Type	Length	Width	Demands
1	3	2	6
2	5	4	3
3	6	3	4
4	7	5	2

Table 2: Example of a paper request with requirements.

There are 3 types of stocks sheets with different sizes, with unlimited supplies:

Type	Width	Length
1	15	10
2	10	8

Table 3: Stock

We have a possible solutions for each kinds of stocks:

Stock type	Rep.	Order no. 1	Order no. 2	Order no. 3	Order no. 4	% Offset
1	1	6	3	3	0	0%
	1	1	0	2	3	2%
2	1	0	4	0	0	0%
	2	1	0	2	1	3.75%
	1	4	0	1	1	3.75%

Table 4: Demonstration of each type of Stock sizes

To solve this problem, we propose a problem modeling based on the Integer Linear Programming:

3.2 Problem Modeling using Integer Linear Programming

Parameters:

- I: Set of demand item types, indexed by i $(1, 2, ..., |I|)$.
- J: Set of stock item types, indexed by j $(1, 2, ..., |J|)$.
- l_i: Length of demand item type i.
- w_i: Width of demand item type i.
- d_i: Demand of demand item type i.
- L_j: Length of stock sheet type j.
- W_j: Width of stock sheet type j.
- R: Set of possible rotations, 0 and 90 degrees, with $R = \{0, 1\}$ (0 is no rotation, 1 is a 90-degree rotation).

Decision Variables:

- x_{ijr}: Integer variable representing the number of items of type i cut from stock sheets of type j with rotation r.
- y_j: Integer variable representing the number of stock sheets of type j used.

Objective Function:

Minimize the total waste:

$$\text{Minimize } \sum_{j \in J}(L_j \cdot W_j \cdot y_j) - \sum_{i \in I}\sum_{j \in J}(x_{ij0} \cdot l_i \cdot w_i + x_{ij1} \cdot w_i \cdot l_i) \tag{1}$$

Constraints:

1. Demand Constraint:
$$\sum_{j \in J}\sum_{r \in R} x_{ijr} \geq d_i \quad \forall i \in I \tag{2}$$
2. Stock Sheet Capacity Constraint:
$$\sum_{i \in I}(x_{ij0} \cdot l_i \cdot w_i + x_{ij1} \cdot w_i \cdot l_i) \leq L_j \cdot W_j \cdot y_j \quad \forall j \in J \tag{3}$$
3. Non-negativity and Integrality:
$$x_{ijr} \geq 0, \text{ integer} \quad \forall i \in I, j \in J, r \in R \tag{4}$$
$$y_j \geq 0, \text{ integer} \quad \forall j \in J \tag{5}$$

4 Problem-solving Algorithms

4.1 Genetic Algorithm

Genetic Algorithms (GAs) are a class of evolutionary algorithms inspired by natural selection. They are well-suited for solving complex optimization problems like the MSS-2DCSP.

4.1.1 Algorithm Process

The essence of genetic algorithms is to encode the optimization function into an array of bits or strings representing chromosomes, and genetic operators operate on the strings and select according to their fitness in order to find a good (or even optimal) solution to the problem in question.

This is usually done by the following process:

1. Encoding the Objective Function

The solution to the problem (i.e., the arrangement of rectangular parts on different stock sizes) is represented as a string of decimal integers. Each chromosome can be divided into sections, where each section represents a specific stock size. Within each section:

- Each integer can represent a rectangular piece.
- The order of integers can represent the order of placement on the stock.

2. Define the Fitness Function or Selection Criteria

The primary objective is to minimize waste. The fitness function evaluates the quality of each solution (chromosome). A common approach is:

$$f = \frac{Area}{Area_1}$$

Which:

- $Area$: The total area of all rectangular parts.
- $Area_1$: The area used in the rectangular plate after the arrangement is completed (i.e., the area below the height contour line of the arrangement diagram).

The larger the fitness value, the better the arrangement effect, that is, the more compact the rectangular parts are arranged on the plate.

3. Create a Population and Individuals

Random Generation: Randomly generate a certain number of decimal integer sequences (chromosomes) as the initial population.

4. Complete the evolution cycle or iteration by evaluating the fitness of all individuals in the population

This is the core of the GA. It involves iteratively improving the population through selection, crossover, and mutation.

4.1 Evaluation:

Decode Chromosomes: For each chromosome in the population, decode the representation into a physical layout of items on the stock sheets.

Calculate Fitness: Calculate the fitness value of each chromosome using the defined fitness function (as described in step 2). This may involve calculating the area of wasted space or the total area of stock used.

Placement Algorithm: You'll need a placement algorithm to determine the actual position of items on the stock sheet during decoding. Common strategies include:

Bottom-Left: Place items starting from the bottom-left corner of the stock sheet.

First-Fit: Place an item in the first available position that can accommodate it.

Best-Fit: Place an item in the position that leaves the least amount of remaining space.

Constraint Handling: During placement, ensure that no items overlap and that all items are placed within the boundaries of the stock sheet. If constraints are violated, either assign a very low fitness value to the chromosome or attempt to repair the solution (e.g., by shifting or rotating items).

4.2 Selection:

Fitness Proportional Selection (Roulette Wheel): Individuals are selected for reproduction with a probability proportional to their fitness. Fitter individuals have a higher chance of being selected.

Tournament Selection: Randomly select a subset of individuals (a "tournament"), and the fittest individual from that subset is selected.

Elitism: A small number of the fittest individuals are directly copied to the next generation to ensure that the best solutions are not lost.

5. Create a new population by performing crossover and mutation

Crossover Operator: Single-point and double-point crossover operators are used to simulate genetic recombination. In single-point crossover, a random position is selected in the parents' chromosomes, and the subsequences after that position are exchanged. In double-point crossover, two random positions are selected, and the subsequence between these two positions is exchanged. The crossover probability is set to 1, meaning all individuals will undergo crossover.

Mutation operator: Position mutation and rotation mutation operators are used to simulate genetic mutation. In position mutation, a random value in the individual's chromosome is changed. In rotation mutation, the rotation angle of a rectangular part is changed. The mutation probabilities for position and rotation mutations are both set to 0.1. The selection operator

Genetic Algorithm

Objective function $f(\boldsymbol{x})$, $\boldsymbol{x} = (x_1, ..., x_d)^T$
Encode the solutions into chromosomes (strings)
Define fitness F (eg, $F \propto f(\boldsymbol{x})$ for maximization)
Generate the initial population
Initialize the probabilities of crossover (p_c) and mutation (p_m)
while ($t <$Max number of generations)
    Generate new solution by crossover and mutation
    Crossover with a crossover probability p_c
    Mutate with a mutation probability p_m
    Accept the new solutions if their fitness increase
    Select the current best for the next generation (elitism)
    Update $t = t + 1$
end while
Decode the results and visualization

selects individuals for the next generation based on their fitness values. In this case, the top M individuals with the highest fitness values are selected.

6. Fitness proportional reproduction

Fitness ratio selection: According to the fitness value of the individual, select the individual to enter the next generation according to a certain probability.

Select the top M: Select the M individuals with the highest fitness value to enter the next generation.

7. Replace the old population and re-iterate the new population

The new population replaces the old population, and the process (steps 4-7) is repeated.

8. Termination

Maximum Generations: The algorithm stops after a predefined number of generations.

Fitness Threshold: The algorithm stops when a solution with a satisfactory fitness value is found.

Convergence: The algorithm stops when the population converges, meaning there is little improvement in fitness over several generations.

Pseudocode for the algorithm:

4.2 Column Generation

The multiple size stock 2D cutting stock problem (MS2DCSP) extends the classic 2D cutting stock problem by allowing for multiple sizes of stock rectangles. This adds significant complexity, as the algorithm must not only determine efficient cutting patterns for each item, but also select which stock rectangles to utilize and in what quantity. This section details a column generation approach tailored to the MS2DCSP.

4.2.1 Master Problem (MP)

The Master Problem (MP) is formulated as an Integer Linear Program (ILP) whose variables represent cutting patterns placed within different stock sizes. It seeks to minimize the total area of stock used.

Sets:

- S: Set of available stock rectangle sizes, indexed by s.
- I: Set of item types to be cut, indexed by i.
- P_s: Set of all feasible cutting patterns for stock rectangle size s, indexed by p. Note: This set is massive and only partially generated as needed.

Parameters:

- L_s, W_s: Length and width of stock rectangle size s.
- l_i, w_i: Length and width of item type i.
- d_i: Demand for item type i.
- a_{ips}: Number of items of type i obtained in cutting pattern p from stock size s.

Decision Variables:

- x_{ps}: Non-negative integer variable representing the number of times cutting pattern p is used on a stock rectangle of size s.

Formulation:

$$\text{Minimize: } \sum_{s \in S} \sum_{p \in P_s} (L_s \times W_s) x_{ps} \text{ (Total area of stock used)}$$

$$\text{Subject to: } \sum_{s \in S} \sum_{p \in P_s} a_{ips} x_{ps} \geq d_i \forall i \in I \text{ (Demand satisfaction for each item)}$$

$$x_{ps} \geq 0 \text{ and integer } \forall s \in S, p \in P_s$$

4.2.2 Restricted Master Problem (RMP)

Due to the exponential size of P_s, the RMP works with a manageable subset of patterns $P'_s \subseteq P_s$.

Initialization: Generate initial feasible patterns, potentially with a simple heuristic or by considering cutting patterns using only a single item type per stock rectangle. This provides a starting feasible solution and initial columns for the RMP.

Iteration:

1. Solve the LP relaxation of the current RMP.
2. Use dual variables (π_i) obtained from the RMP solution to generate new patterns by solving the slave problem.
3. If new patterns with negative reduced cost are found, add them to P'_s and repeat from Step 1.
4. If no negative reduced cost patterns are found, the current RMP solution is optimal for the LP relaxation of the MP.

4.2.3 Slave Problem (SP)

For each stock rectangle size s, the slave problem (SP) aims to find a cutting pattern p with minimal reduced cost. It can be formulated as a 2D Knapsack problem with two-staged guillotine cuts. Several approaches are possible:

- **Direct Integer Programming approach using M0 model.** A variant of model M0 detailed previously adapted for different stock sheet sizes.
- **Heuristic Pattern Generation** This method can prove significantly faster for the following stages, particularly after the first iteration which can be very computationally expensive with an ILP based sub-problem. A heuristic variant can quickly determine a new useful cut to include, if it is deemed worth expanding the columns being evaluated via another expensive exact model optimization step.

These approaches use the dual values obtained for items (π_i) in the current iteration as "profits". When rotation is considered, copies of items in alternate configurations would be handled explicitly by increasing the number of items $|I|$ considered in I for which solutions may potentially require rotations, but no implicit orthogonal rotations must be accounted for in solutions to models generated without rotated pattern types. A cutting pattern resulting in a solution from an individual SP would include both a rectangle type from s as well as a selected collection of cut rectangles as before and potentially rotational state (or duplicate items pre-rotated and fixed).

The interplay between the RMP and the various SPs, for different size stocks, iteratively improves the solution. Once the LP relaxed optimum of MP is identified, integer strategies are necessary to derive the final solution for the overall MS2DCSP problem as presented previously with any required 2-stage orthogonal rotational states explicitly tracked as part of generated

cutting patterns via their selected cut types from rotated item configurations generated at that initial column inclusion phase using heuristics of model calculations where possible as deemed worthwhile when computationally advantageous before potentially invoking higher quality cost exact sub-problem variants when required by considering them as separate patterns, thus forming their respective associated SP model constraint problem solutions and associated decision variables.

Pseudocode for Column Generation:

```
repeat
    Start with a set of initial patterns (m) and solve the master problem.
    Find the multipliers corresponding to the demand constraints: get v\_i.
    Solve the subproblem (knapsack) and obtain a new cutting pattern.
until the subproblem has a negative objective value;
```

5 Heuristic Approaches Using Greedy Algorithm and Bottom-left Algorithm

1. Objective:
Greedy Algorithm and Bottom-left Algorithm aim to minimize resource utilization while maximizing placement efficiency. The objective can be expressed as:

$$\text{Minimize: } \sum_{b=1}^{N_b} c_b$$

Where:

- N_b: Number of bins or sheets used.
- c_b: Cost associated with each bin or sheet (e.g., area, material, or operational cost).

2. Constraints:

5.1 Placement Constraints

1. **Item Fit within the Bin/Sheet:**

$$x_j + w_j \leq W, \quad y_j + h_j \leq H$$

Where:

- W, H: Width and height of the bin or sheet.
- x_j, y_j: Bottom-left corner coordinates of item j.
- w_j, h_j: Width and height of item j.

2. **Non-Overlap Condition:**

$$(x_j \geq x_k + w_k) \vee (x_j + w_j \leq x_k) \vee (y_j \geq y_k + h_k) \vee (y_j + h_j \leq y_k) \quad \forall j \neq k$$

- Left of k $(x_j \ \geq x_k + w_k)$: Item j's left edge is to the right of item k's right edge.
- Right of k $(x_j + w_j \leq x_k)$: Item j's right edge is to the left of item k's left edge.
- Above k $(y_j \geq y_k + h_k)$: Item j's bottom edge is above item k's top edge.
- Below k : $(y_j + h_j \leq y_k)$: Item k's top edge is below item k's bottom edge.

Demand and Capacity Constraints (Greedy Specific)

1. **Demand Constraint:**

$$\sum_p \sum_r a_{pf} \cdot x_{pr} \geq d_f, \quad \forall f$$

Where:

- d_f: Demand for the final product f.
- a_{pf}: Number of final pieces f obtained from cutting pattern p.
- x_{pr}: Number of raw materials r cut using pattern p.

2. **Capacity Constraint:**

$$\sum_p x_{pr} \leq k_r, \quad \forall r$$

Where:

- k_r: Available capacity of raw material r.

Placement Order (Bottom Left Specific)

1. **Bottom-Left Rule:**
   - Minimize y-coordinate first (lowest position).
   - For the same y-coordinate, minimize x-coordinate.
2. **Item Placement:** Each item must be placed exactly once:

$$\sum_{b=1}^{N_b} \delta_{j,b} = 1, \quad \forall j$$

Where:

- $\delta_{j,b} = 1$ if item j is placed in bin b, otherwise 0.

3. Variables:

- x_j, y_j: Coordinates of the bottom-left corner of item j.
- w_j, h_j: Width and height of item j.
- x_{pr}: Number of raw materials r cut using pattern p.
- $\delta_{j,b}$: Binary decision variable indicating whether item j is placed in bin b.

6 Evaluation

We evaluate some heuristics like Genetic Algorithm, Column Generation in some Case Studies, and the speed of the Optimized Greedy Algorithm with the Bottom Left Algorithm.

These values are calculated using Python.

6.1 Evaluation between Algorithms

This section evaluates three different policies for solving the cutting stock problem: the Optimized Greedy Policy, the Bottom-Left Policy, and a Random Policy. The Optimized Greedy Policy, an enhancement over the standard Greedy approach, incorporates sorting of both products and stock materials to improve efficiency, whereas the Bottom-left (BL) policy is a heuristic algorithm commonly used, it's known for its simplicity and relatively good performance in generating feasible solutions, especially when dealing with rectangular shapes.

The result can be seen as the table below:

Policy	Number of Stock	Seed	Average Time (s)	Avg. Trim Loss
Optimized Greedy Policy	16	4	131.7421	0.074
Bottom-left Policy	25	4	47.1067	0.257
Random Policy	97	4	0.3857	0.782

Our analysis reveals the following key observations:

- **Execution Time:** The Random Policy exhibits the shortest average execution time. This is expected as it does not involve any optimization of trim loss. Conversely, the Optimized Greedy Policy requires the longest processing time, primarily due to the overhead associated with sorting. The Bottom-Left Policy offers a balance between time and trim loss.
- **Trim Loss:** Optimized Greedy Policies achieve a relatively low average trim loss percentages, whereas the Bottom-left Policy has higher trim loss percentage. The Random Policy, as anticipated, results in significantly higher trim loss because of a lack of optimization strategy. This highlights the trade-off between computational effort and the efficiency of material utilization when comparing different policies.

6.2 Evaluate Column Generation with other outside models

For having an another sight of Column Generation, we will having a case study, evaluate between the Column Generation and a independent web-based utility called OptiCutter.

6.2.1 Case study: Paper Cutting Problem

A paper manufacturing plant needs to cut A0 size paper (841mm x 1189mm) into smaller sizes to fulfill an order. The order specifies the following paper sizes and quantities:

- 50 sheets (297 x 420 mm)
- 120 sheets (210 x 297 mm)
- 80 sheets (148 x 210 mm)

The goal is to minimize the waste of the stocks.

We will have the table of the order demands:

Type	Length (mm)	Width (mm)	Quan.
1	420	297	50
2	297	210	120
3	210	148	80

Table 5: The sizes and demands of the order.

The following table is the result of the outside software:

Layout	Type 1	Type 2	Type 3	Repetition	Total Offset (%)
1	8	3	5	6	
2	7	5	5	1	8.1%
3	0	7	27	1	
4	0	18	4	5	

Table 6: Cutting Plan of OptiCutter

This is the layout results from Column Generation:

Layout	Type 1	Type 2	Type 3	Repetition	Total Offset (%)
1	4	4	0	0	
2	0	16	0	5	
3	0	0	25	0	15.53%
4	3	3	8	7	
5	4	3	3	8	

Table 7: Cutting Plan from Column Generation

From this case study, both OptiCutter and Column Generation provide feasible solutions that meet the order demands. However, OptiCutter demonstrates a superior outcome in this particular scenario by achieving a significantly lower waste percentage (8.1% compared to 15.53%). This highlights the importance of choosing the right optimization tool or algorithm based on the specific problem and its constraints.

7 Conclusion

7.1 Results

- Successfully modeled the problem as an integer linear program.
- Implemented and understood the application of Bottom-left and Greedy Algorithm in solving the multiple stock two-dimensional cutting stock problem.
- Understood the application, advantages, and disadvantages of the algorithms in the multiple stock two-dimensional cutting stock problem.
- Analyzed the implemented algorithm on a dataset.
- Compared the performance of different models with each other and with independent software for solving the existing problem.
- Collected data to solve the problem.

7.2 Limitations

- The implemented model still needs significant improvement in terms of optimizing the number of waste material.
- The runtime of the Optimized Greedy algorithm has not been optimized.

7.3 Future Work

- Implement Genetic Algorithms for solving the problem.
- Develop more models to solve the two-dimensional cutting stock problem quickly and efficiently.

References

[1] Salem, Ossama & Shahin, A. & Khalifa, Yaser. (2007). *Minimizing Cutting Wastes of Reinforcement Steel Bars Using Genetic Algorithms and Integer Programming Models*. Journal of Construction Engineering and Management-asce - J CONSTR ENG MANAGE-ASCE. 133. 10.1061/(ASCE)0733-9364(2007)133:12(982).

[2] Mellouli, Ahmed & Dammak, Abdelaziz. (2008). *An Algorithm for the Two-Dimensional Cutting-Stock Problem Based on a Pattern Generation Procedure*. Information and Management Sciences. 19. 201-218.

[3] Silva, E., Alvelos, F., & de Carvalho, J. M. V. (2010). *An integer programming model for two- and three-stage two-dimensional cutting stock problems.* European Journal of Operational Research, 205(3), 699–708. https://doi.org/10.1016/j.ejor.2010.01.018

[4] Bökler, F., Chimani, M., & Wagner, M. H. (2022). *On the rectangular knapsack problem. Mathematical Methods of Operations Research.* https://doi.org/10.1007/s00186-022-00788-8

[5] Lo Valvo, E. (2017). Meta-heuristic algorithms for nesting problem of rectangular pieces. Procedia Engineering, 183, 291–296. https://doi.org/10.1016/j.proeng.2017.04.038

[6] Manuel Iori, Vinícius L. de Lima, Silvano Martello, Flávio K. Miyazawa, Michele Monaci, Exact solution techniques for two-dimensional cutting and packing, European Journal of Operational Research, Volume 289, Issue 2, 2021, Pages 399-415, ISSN 0377-2217, https://doi.org/10.1016/j.ejor.2020.06.050.