Vietnam National University, Ho Chi Minh City
University of Technology
Faculty of Computer Science and Engineering

MATHEMATICAL MODELING (CO2011)

Assignment (Semester: 241, Duration: 04 weeks)

"Cutting Stock Problem"

Instructor(s):
Nguyen Van Minh Man, Mahidol University
Nguyen An Khuong, CSE-HCMUT
Le Hong Trang, CSE-HCMUT
Mai Xuan Toan, CSE-HCMUT
Tran Hong Tai, CSE-HCMUT
Nguyen Quang Duc, CSE-HCMUT

Student(s):
Nguyen Huu Phuc - 2352938 (Group CC05 - Team 03, **Leader**)
Tran Bao Phuc Long - 2352703 (Group CC05 - Team 03)
Nguyen Thanh Hieu - 2352331 (Group CC05 - Team 03)
Truong Gia Ky Nam - 2352787 (Group CC05 - Team 03)
Chau Anh Nhat - 2352856 (Group CC05 - Team 03)

Ho Chi Minh City, December 2024

Abstract

The Cuttting Stock Problem (CSP) is a well optimization challenge in Operations Research and Industrial Engineering. It involves minimizing the waste material while cutting raw stock sheets into smaller pieces to satisfy demand, a challenge commonly encountered in manufacturing and logistics industries.

This report aims to present a novel approach for solving the 2D Cutting Stock Problem using lazy stock initialization and column generation techniques. The proposed method begins with an iterative lazy initialization of stock sheets, where stock is dynamically introduced into the solution process only when required, reducing the initial problem size and enhancing computational efficiency. A column generation framework is then employed to iteratively solve a restricted master problem (RMP) and a subproblem. The RMP determines the optimal combination of cutting patterns for the current stock configuration, while the subproblem generates new, cost-effective patterns using a knapsack-based optimization approach.

Member List & Workload

No.	Fullname	Student ID	Contributions	% Done
1	Nguyen Huu Phuc	2352938	Modeling, Column generation theory	100%
2	Truong Gia Ky Nam	2352787	Implement Modeling, Column generation	100%
3	Tran Phuc Bao Long	2352703	Implement Lazy Initialization, Pattern generation	100%
4	Chau Anh Nhat	2352856	Report consolidate, Case study	100%
5	Nguyen Thanh Hieu	2352331	Introduction, acknowledge, Pseudo code	100%

Contents

1 Introduction

The Cutting Stock Problem (CSP) is a classic optimization challenge that arises in various industries. Introduced by P.C.Gilmore and R.E.Gomory in the 1960s, CSP has evolved significantly, impacting businesses such as manufacturing, textiles, and packaging, where material efficiency directly influences costs. The core objective of CSP is to determine how to cut larger stock materials into smaller items to meet demand while minimizing waste and maximizing the use of available resources. Due to its combinatorial nature and the exponential growth of potential solutions, the CSP is considered an NP-hard problem, requiring specialized algorithms and methods for practical solutions.

1.1 Historical background and evolution

The initial work on the CSP focused on one-dimensional variants, where materials like paper rolls were cut into narrower strips. Gilmore and Gomory developed mathematical programming models and solution methodologies that laid the groundwork for modern research. Their approach utilized linear programming with column generation, which proved to be a significant step toward solving large-scale instances of the problem efficiently.

Over time, research expanded to two-dimensional and multi-dimensional variants, which are more applicable to industries dealing with stocks of material rather than simple linear rolls. The Two-Dimensional Cutting Stock Problem (2D CSP) emerged as an important field of study, with methods being adapted to handle complexities related to layout, orientation, and multiple constraints.

1.2 Significance in Various Industries

The CSP holds great importance in industries that prioritize material efficiency and cost reduction. For instance:

- **Manufacturing:** Metalworking, glass cutting, and wood processing plants need to optimize their cutting patterns to minimize scrap and meet production quotas.
- **Textiles:** The fashion and upholstery sectors face challenges in cutting fabrics into specific shapes with minimal waste.
- **Printing and Packaging:** Paper manufacturers and packaging companies use CSP solutions to optimize the cutting of large stocks into smaller items for printed materials or packaging designs.
- **Logistics and Construction:** Industries that deal with materials like plastic or insulation stocks also benefit from optimized cutting stock solutions. An effective CSP solutions improve operational efficiency, lower material costs, and support sustainability initiatives by reducing waste.

1.3 Mathematical Models Used in CSP

Over the years, various mathematical models have been developed to represent and solve the CSP. These models differ based on the nature of the problem (e.g., one-dimensional or multi-dimensional) and the complexity of constraints involved. Some of the common models include:

- **Linear Programming (LP) and Integer Linear Programming (ILP):** Core methods for exact solutions, employing column generation to optimize cutting patterns by solving sub-problems iteratively.
- **Knapsack-Based Models:** Represent CSP as an extension of the knapsack problem, focusing on maximizing the use of stock material within capacity limits.
- **Mixed-Integer Programming (MIP):** Used for problems with additional constraints, such as minimizing the number of distinct cutting patterns or incorporating production costs.
- **Greedy Heuristics and Constructive Methods:** These offer quick, practical solutions, generating an initial arrangement that can be refined using more advanced optimization methods. Common examples include First-Fit Decreasing (FFD) and Best-Fit Decreasing (BFD) heuristics.

1.4 Goal

The 2-dimensional cutting stock problem (2D CSP) is a type of optimization problem where the goal is to cut smaller rectangular items from larger rectangular stocks of material in a way that minimizes the area of used stocks if a fixed number of stocks are used. It is commonly encountered in manufacturing and production industries where efficient material usage is essential for cost savings.

2 Acknowledge

2.1 Categorizing the Problem

The 2-dimensional cutting stock problem can be differentiate into many kinds of problem based on these factors:

- **Types of Cutting Patterns:**
  - **Guillotine Cuts:** Each cut must go from one edge of the material to the other, simplifying the cutting process. Guillotine cutting is often preferred because it's easier to automate and more practical in large-scale manufacturing.
  - **Non-Guillotine Cuts:** Cuts don't have to span the entire length or width, which can lead to more efficient use of material but is usually harder to implement due to increased cutting complexity.
- **Types of Stocks:** The problem includes one or multiple types of stocks, each with either a finite limit on the quantity or an infinite limit.
- **Orientation Constraints:** The stock items can be freely rotated in any directions or must remain in a specific orientation depends on the stock's properties
- **Shape of Stock:** The stock can be rectangle, which will be easy to solve and cut or it can have arbitrary shapes which requires a more complex approach to solve
- **Common problems:** Based on the final goal of the problem, we can create many variation of this problem. We mainly deal with these 4 types:
  - **2-dimensional strip packing problem:** Given a single stock B with fixed width W and infinite height, this problem's demand is to fit a packing of items that use minimal height.
  - **2-dimensional knapsack problem:** Given a value V to each item in our stock S, each item's value can be different, the goal is to fit the most items of S in a single stock B that have the highest sum of value.
  - **2-dimensional stock stacking problem:** We have an infinite amount of identical stock B that have height H and width W along with a finite amount of item called set I. This problem wants us to divide set I into minimum subsets such that each subset can be put inside a stock B, this can be rewritten as this problem wants us to use the least number of stock B to put all of set I's items inside.
  - **2-dimensional orthogonal packing problem:** This problem simply asks if there is a pattern to pack a given set of items I into a single stock B or a set of stock B.

2.2 Main Problem Definition

- **Input Stock Material:** You start with a set of multi-sized stocks for each stock there is limit for quantity, which serves as the stock from which smaller items are cut. In this report, we consider 2 stages guillotine cut and we allow trimming (a third stage cut can be used to separate a rectangle from a waste area)
- **Demanded Items:** There's a set of smaller item, each with specific size and demand (rotation is allowed), that must be cut from the stocks.
- **Goal:** The objective is to cut the demanded items from the stock in a way that minimizes the area of used stocks if a fixed number of stocks are used.

2.3 Types of Constraints

Several constraints typically govern the solution:

- **Constraint for the demand of each item type:** Each item demanded must be produced in the required quantity.
- **Constraints for the quantity of each stock type:** Do not use over the limit of the quantity of each stock type

2.4 Difficulties in Solving 2D CSP

- The items have different dimensions, some long and narrow, others square or almost square. This variety complicates the arrangement, as items cannot easily "tessellate" or fit together like a puzzle without leaving gaps.
- Large items reduce flexibility in the arrangement, limiting the space left for smaller items.
- The items cannot overlap on the stock; each item must occupy a unique position.
- Since the stock is a fixed size, there's a risk of not fitting all items if the layout isn't planned carefully.
- In some configurations, certain items might not fit, forcing the need to use additional stocks

2.5 Real-World Difficulties in Solving 2D CSP

- **Computational Complexity:** Finding the optimal layout among all possible arrangements is computationally intensive, especially with larger stocks and a wide variety of parts.
- **Material Costs:** High-value materials add pressure to reduce waste as much as possible. An inefficient layout could significantly increase material costs.
- **Production Flexibility:** Factories often handle multiple types of materials and need to switch between layouts quickly. This flexibility is hard to balance with the need for material efficiency.
- **Cutting Constraints:** Certain materials or machines may have cutting constraints (minimum cut width, grain direction for wood or textiles), which restrict how items can be oriented and add further complexity.
- **Time constraints:** Low time budget might prevent global optimization methods, so near-optimal or heuristic approaches are often required, risking suboptimal results.

2.6 General Solutions

- **Exact Finding Methods:** Linear Programming(LP), Mixed-Integer Linear Programming (MILP)
- **Heuristics and Metaheuristics Methods:** Greedy algorithms, First Fit and Best Fit strategies, Genetics algorithms, simulated annealing and tabu search
- **Hybrid Approaches:** Combining heuristics and optimization and decomposition techniques

3 Algorithm

3.1 Lazy Initialization

The lazy stock initialization heuristic in the 2-dimensional cutting stock problem is a strategy that defers the evaluation of item combinations and strip configurations until they are needed during the optimization process. Instead of precomputing all possible item placements for each strip, the algorithm dynamically evaluates configurations only when required, reducing computational overhead and memory usage. This approach allows the algorithm to focus on the most promising item combinations, initializing and placing

items as necessary based on the current iteration. As strips are filled, leftover space is addressed by lazily evaluating potential fits for additional items. While this heuristic helps save resources and provides flexibility, it may lead to increased runtime when a deferred configuration is eventually needed and can complicate implementation due to the need to manage on-the-fly evaluations. Overall, lazy stock initialization improves scalability and adaptability in solving large-scale cutting stock problems.

```
Initialization
    Define bucket_size = 10 // This means each bucket will cover a size range of 10 units of stock_w and stock_h
    Create empty dictionary stock_buckets = {} // Key in bucket is the bucket ranges
    Set best_stock_idx = -1, best_position = None, best_prod_size = [0,0]
    Sort items in descending order of area on the first time running // Sort stock in ascending order of area (optional)
Group stock into buckets:
    For each stock:
        Calculate bucket key = dimensions // bucket_size
            Assign stock to appropriate bucket
Iteration
    For each item:
        If item quantity > 0:
            Get size of item
            Set minimum_waste_percent = inf
            For each candidate stock: // From the grouped buckets, retrieve stocks that are large enough to potentially fit the item using keys
                Check if item can be placed in the stock by iterate through position in the stock
                    If can, calculate the waste percentage as the unused area in the stock
                    Update the best_stock_idx, position and prod_size if has lower waste_percentage
Return result
    If suitable placement is found:
        Reduce the quantity by 1
        Return prod_idx, stock_idx and position
    Else
        Return stock_idx = -1, size = [0, 0], position = None
```

Advantages:

- **Reduced Initial Overhead:** By not precomputing all product layouts in advance, the algorithm decreases upfront computational time and resource consumption.
- **Lower Memory Usage:** Since it avoids storing all potential configurations, the approach conserves memory, which is especially beneficial for large-scale problems.
- **Focus on Promising Configurations:** The method only evaluates configurations when needed, allowing it to concentrate on those most likely to improve the solution.
- **Flexibility:** As the solution space evolves, the algorithm can adaptively evaluate new item placements without being constrained by predetermined layouts.
- **Scalability:** By reducing initial computation and memory demands, the method can handle larger problem instances more efficiently.

Disadvantages:

- **Potential Increased Runtime on Demand:** Delaying the evaluation of certain configurations may lead to longer runtimes when these configurations are eventually required.
- **Implementation Complexity:** Managing on-the-fly evaluation of configurations is more intricate, increasing the complexity of coding and maintenance.

- **Uncertain Runtime Behavior:** Because configurations are evaluated as needed, predicting the overall runtime is more challenging.
- **Need for Effective Search Strategies:** To realize the full benefits of lazy evaluation, the search strategy must be well-designed to avoid excessive deferred computations.

3.2 Column Generation

Column generation is a precise and iterative optimization technique specifically designed to solve the cutting stock problem efficiently, particularly when the master problem is exponentially large. The process begins by establishing a restricted master problem (RMP), which includes only a manageable subset of patterns derived from initial heuristic methods to avoid the complexity of the full exponential master problem. This RMP is then solved as a linear programming problem to obtain a fractional solution. The dual variables from this solution are used to identify new patterns with negative reduced costs by solving a two-dimensional knapsack problem for each stock class, ensuring that only the most promising patterns are considered. In particular, the algorithm takes all patterns with negative reduced costs, ensuring comprehensive inclusion of beneficial configurations. Once these new patterns are generated, they are added to the RMP, and the master problem is re-optimized. This cycle of solving the RMP, generating new patterns using the dual variables, and re-optimizing continues iteratively. The algorithm terminates when no additional patterns with negative reduced costs can be found, indicating that the optimal solution to the cutting stock problem has been reached. By restricting the master problem to a subset of patterns and systematically expanding it with beneficial patterns, column generation effectively manages the exponential complexity, ensuring an exact and optimal solution.

```
Input:
    items, demands, stocks

Output:
    optimal_patterns, total_stocks_used

Initialization
    patterns <- generate_initial_patterns(items, stocks)  # Generate feasible initial patterns
    RMP <- setup_rmp(patterns, demands)  # Set up restricted master problem (RMP)
    dual_prices <- []  # Initialize dual prices list

Solve RMP using column generation
    while True:
        solve_linear_program(RMP)  # Solve linear relaxation of RMP
        dual_prices <- extract_dual_prices(RMP)  # Get dual prices from solved RMP

        # Generate new patterns using dual prices
        new_patterns <- solve_knapsack(items, stocks, dual_prices)  # Solve knapsack problem
        if reduced_cost(new_patterns, dual_prices) is greater than or equal to 0:  # Check if new pattern
        ↪  improves solution
            break  # Stop if no new patterns can improve the objective
        add_patterns_to_rmp(RMP, new_patterns)  # Add new patterns to RMP if improvement is possible

    Extract Final Solution by solving RMP as an integer linear program
    final_solution <- solve_integer_linear_program(RMP)

Format Output
    optimal_patterns <- []  # Initialize list to store optimal patterns
    total_stocks_used <- 0  # Initialize counter for total stocks used
    for pattern in final_solution:  # Iterate over final solution patterns
        if pattern.usage > 0:  # Only include patterns used in the solution
            optimal_stocks.append(pattern)  # Add pattern to optimal stocks
            total_stocks_used += pattern.usage  # Accumulate total usage

```

```
Return Result
    print(total_stocks_used)  # Optionally print the result for user reference
    return optimal_patterns  # Return list of optimal stocks
```

Subproblems in column generation:

- **Initial heuristics:** This technique provides a straightforward method for generating feasible solutions to the two-dimensional cutting stock problem. One widely used heuristic is the strip packing approach, where items are sorted by dimensions (width and length) and then packed into strips that align with guillotine cutting constraints. In this process:
  - Items are first sorted in descending order by width, and ties are broken by length.
  - Strips are created to fit as many items as possible without exceeding the dimensions of the stock.
  - A greedy filling approach is used to fill any remaining space in the strips with smaller items.

```
For each item class in item set
        Create a rotated version of that class with the same quantity
        Insert it to the item class set

Sort the new items set by height then by width (descending)

For each item class in item set:
        While item demand is not meet:
                Find the smallest stock class that a single stock can fit all or the most of item
                Insert item to stock by the following predefined order
                While remaining stock height is available:
                        Initialize a strip with height equal to height of the first inserted item
                        Fill the strip with items that have the class of the first item
                        When no additional item of that class can fit, look for other item class that can fill in the remaining space until no item can be inserted.
```

- **Pattern Generation (Knapsack problem):** We introduce a heuristic pattern generation, which does not guarantee exact optimality but provide efficiency in solving large problems (increasing by type of stock and item). This algorithm for pattern generation in solving the 2-dimensional cutting stock problem involves several key steps:
  - Determine the possible strip layouts based on the item types, considering both the original and rotated orientations of each item.
  - Evaluate how many items of each item type can fit into each strip, ensuring minimal waste.
  - After the initial item allocation, optimize the strip's capacity by:
    - Checking for leftover space.
    - Attempting to fill leftover space with additional items from other item types.
    - Exploring all feasible combinations for filling.
  - Store these valid configurations in a "strip list" for later evaluation.
  - Calculate the total profit for each combination using the profit margins of the items.
  - Check if the combination meets the demand for the required number of items.
  - Identify and return the most profitable configuration that satisfies the demand and is feasible.

```
Input
    dual_prods, stock_type

```

```
Output
    max_result

Initialize
    init_list <- []
    top = 6
    stock_w, stock_h <- stock_type.width, stock_type.height
    item_dims <- extract_dimensions(items)

Generate Strips
    for height in item_dims.heights:
        if height > stock_h: continue
        counts <- [stock_w // w for w in item_dims.widths if w  stock_w and height  height]
        init_list.append({height, counts})

Sort Items
    sorted_items <- sort(clone(items), dual_prods, descending=True)

Process Strips
    strip_list <- []
    for strip in init_list:
        for each prod_type in strip:
            Divide into n array for n item
            prod[i] = itemCount[i], else = 0
            for item in sorted_items:
                if can_fit(strip, item):
                    place_item (strip, item)
            strip_list.append(strip)

Create matrices for optimization
    top_strips, h_strips, h_stock, min_array <- [], [], [], []
    for height in item_dims.heights:
            current_strips <- strips height == height
        sort current_strips by profit
        for strip in top 6 current_strips by profit:
            top_strips.append(top 6 of current strips)
            h_strips = strip height
            h_stock = stock_height
            min_array = max number of strip can use
MILP
    Pass top_strips, h_strips, h_stock and min_array for milp solving
    Return pattern of strips
    Combine into stock
    return pattern.

Return
    print(max_profit)
    return max_result
```

Advantages:

- **Reduced Computational Complexity:** Heuristic column generation focuses on generating patterns through approximate methods rather than solving the exact knapsack problem for each stock class. This significantly reduces computational time and resources, making it suitable for large-scale problems.
- **Scalability:** By simplifying the subproblem and focusing on heuristic techniques, the approach can handle problems with large numbers of variables or stock classes that would be computationally

expensive to solve exactly.

- **Faster Convergence:** Heuristics often produce good patterns quickly, allowing the algorithm to progress without spending excessive time finding the mathematically optimal pattern at every iteration.
- **Flexibility in Implementation:** Heuristic methods can be tailored to specific problem structures, leveraging domain knowledge to generate patterns efficiently.
- **Practical Solutions:** Heuristic column generation often provides near-optimal solutions that are satisfactory for practical applications, especially when slight deviations from optimality are acceptable.

Disadvantages:

- **Suboptimal Solutions:** Since heuristics do not guarantee optimality, the final solution may not be the true optimal solution to the cutting stock problem.
- **Dependence on Heuristic Quality:** The effectiveness of the approach depends heavily on the quality of the heuristics used for generating patterns. Poorly designed heuristics can lead to slow convergence or exclude beneficial patterns.
- **Potential for Missed Patterns:** Heuristics may fail to identify certain promising patterns with negative reduced costs, leading to a less comprehensive exploration of the solution space.
- **Less General Applicability:** Heuristic column generation techniques may need significant adaptation for different problem instances or constraints, limiting their generalizability.
- **Difficulty in Assessing Optimality:** Since heuristic methods do not guarantee finding all patterns with negative reduced costs, it may be challenging to determine when the solution is close enough to the true optimal solution.

4 Modeling

The two-dimensional Cutting Stock Problem involves cutting a specified quantity of various stock materials into desired items in a manner that satisfies all demand requirements while minimizing the total area of stocks utilized. This optimization ensures efficient use of resources and reduces material waste in the manufacturing process.

Formally, we are given m classes of rectangular items, each item class i $(i = 1, \ldots, m)$ having a demand D_i, and n classes of rectangular stocks, each stock class j $(j = 1, \ldots, n)$ having an limited number of copies. Each item of class i $(i = 1, \ldots, m)$ has dimensions (l_i, w_i), where l_i and w_i are, respectively, the length and the width of the item (orthogonal rotation is allowed). In this context, we consider "=" because of the exact demand the problem requiring (instead of "$\geq$", which is appropriate in real-life as the item may exceed the demand leading to minimizing trim loss and raising profit). Each stock of class j $(j = 1, \ldots, n)$ has dimensions (L_j, W_j), where L_j and W_j are, respectively, the length and the width of the stock. All the input data are assumed to have positive integer values. We define the cutting patterns (simply denoted as patterns in the following) to obtain all the requested items in the specified demand by minimizing the total area of the used stocks.

We model the problem as a linear programming formulation as follows:

$$\text{Min} \quad \sum_{j=1}^{n} \sum_{p_j \in P_j} c_{p_j} \cdot x_{p_j} \tag{1}$$

$$\text{Subject to} \quad \sum_{j=1}^{m} \sum_{p_j \in P_j} C_{p_j}^{i} \cdot x_{p_j} = D_i \quad (i = 1, 2, ..., m) \tag{2}$$

$$\sum_{p_j \in P_j} x_{p_j} \leq S_j \quad (j = 1, 2, ..., n) \tag{3}$$

$$x_{p_j} \in \mathbb{Z}^+ \quad (j = 1, 2, ..., n, p_j \in P_j) \tag{4}$$

List of Symbols:

x_{p_j}	Number of pattern p_j used
c_j	Cost of each pattern p_j (known as the area of the stock that pattern p_j uses)
P_j	Set of possible patterns (set of p_j) of stock j
$C_{p_j}^{i}$	Number of item class i in pattern p_j
S_j	Number of available stocks of stock type j
(1) Objective function:	Minimize the area of stock used
(2) Demand constraint:	Ensure that the required quantities for each class item are fully met
(3) Stock availability constraint:	Ensure the use of stocks does not exceed its type stock limit
(4) Integrality constraint:	All values of x_{p_j} must be positive integers

By dropping the integrality requirement for the variables, imposed by constraints (4), we obtain the Linear Programming (LP) Relaxation of model (1)–(4) defined by (1), (2), (3) and:

$$x_{p_j} \geq 0 \quad (j = 1, 2, ..., n, p_j \in P_j) \tag{5}$$

Model (1), (2), (3) and (5), called **Master Problem (MP)**(or the primal problem), needs column generation techniques to be efficiently managed. It has exponentially many variables, which cannot be explicitly generated for large-size instances. Instead of the original Master Problem, a **Restricted Master Problem (RMP)** is considered which is initialized and solved with a subset of the exponentially many variables x_{p_j} ; $p_j \in P_j$; j = 1;...;n. In particular, RMP is initialized with the variables (columns) corresponding to a feasible solution computed through the initial heuristic algorithm. Given a solution to RMP, new variables, needed to optimally solve MP, are obtained by separating the following dual constraints from the dual problem below:

$$\text{Max} \quad \sum_{i=1}^{m} D_i \alpha_i + \sum_{j=1}^{n} S_j \beta_j \tag{6}$$

$$\text{Subject to} \quad \sum_{i=1}^{m} C_{p_j}^{i} \alpha_i + \beta_j \leq c_{p_j} \quad \text{for each pattern } p_j \tag{7}$$

$$\alpha_i, \beta_j \geq 0 \tag{8}$$

List of Symbols:

α_i	Dual variable for demand constraint
β_j	Dual variable for stock availability constraint

Reduced cost for each pattern p_j :

$$r_p = c_{p_j} - \left(\sum_{i=1}^{m} C_{p_j}^{i} \alpha_i + \beta_j \right) \tag{9}$$

In order to generate new pattern (column) for optimizing the problem or to check optimality, we need to solve a Knapsack problem (pricing problem) to determine new pattern, which reduced cost < 0, to put in RMP. If there is no negative reduced cost then the problem has reached optimality and need to solve for integrality solution. Below is the formulation for the Knapsack problem:

$$\text{Max} \quad \sum_{i=1}^{m} C_{p_j}^i \alpha_i \tag{10}$$

$$\text{Subject to} \quad \sum_{i=1}^{m} c_i C_{p_j}^i \leq c_{p_j} \tag{11}$$

$$C_{p_j}^i \leq D_i \quad (i = 1, 2, 3, ..., m) \tag{12}$$

$$C_{p_j}^i \in \mathbb{Z}^+ \tag{13}$$

List of Symbols:

c_i	Cost of each class i item (known as the area of item class i)
(10) Objective function:	Maximize the profit of the pattern for a specific type of stock
(11) Size constraint:	The area of items in stock must not exceed the stock size
(12) Demand constraint:	The number of items for each class in stock must not exceed their demand
(13) Integrality constraint:	All values of $C_{p_j}^i$ must be positive integers

Summary for Modeling

To solve to optimality MP given by model (1), (2), (3) and (5), a classical column generation scheme is applied. In detail, RMP is initialized with the columns corresponding to a heuristic solution. Then the current RMP is solved to optimality, thus obtaining a vector of dual variables α^* . A Knapsack problem SP is defined for every stock class j (j = 1, . . . , n). Then the Knapsack problem are solved with pattern generation scheme and new columns with negative reduced cost are added to current RMP. If no column with negative reduced cost is found, the current solution (possibly fractional) of RMP is optimal for MP. Finally, we solve the MILP model to obtain integer solution

5 Analyze Result

To evaluate our model, the initial cutting plans were carried out using the standalone software optiCutter and CutList Optimizer. Notably, optiCutter is limited to a maximum of 5 stock types. Additionally, in the comparison section below, we include a comparison with initial patterns used in column generation to demonstrate the optimization achieved by our approach. These same cutting plans were then implemented using our model to compare its level of effectiveness.

5.1 General Assessment

This report will be evaluate by these indexes :

- **Stock Area Used:** is the cumulative area of raw material stocks employed to fulfill the cutting requirements in the 2D Cutting Stock Problem
- **Area Filled Ratio:** is the ratio of the Stock Area Used to the Total Stock Area, indicating how much of the available material is successfully employed in the cutting process.

$$Area\ Filled\ Ratio = \frac{Stock\ Area\ Used}{Total\ Stock\ Area}$$

- **Filled Ratio:** is the ratio of the number of stocks used to the total number of stocks, indicating the proportion of available stocks that are successfully utilized in the process.

$$Filled\ Ratio = \frac{Number\ of\ Stocks\ Used}{Total\ Number\ of\ Stocks}$$

- **Trim Loss:** is the average ratio of the unused area to the total area of each stock after cutting the required items. It indicates how much material is wasted during the cutting process, with a lower trim loss representing higher material efficiency.

$$Trim\ Loss = \frac{1}{n}\sum_{i=1}^{n}\frac{Stock\ Area_i - Items\ Area\ on\ Stock_i}{Stock\ Area_i}$$

Note: n is the number of used stocks.

- **Time:** is the average computational duration needed to solve the 2D Cutting Stock Problem across all problem instances. It provides an indicator of the algorithm's performance, where a lower time signifies a more efficient and faster-solving method. Time represents only the time required for computation and does not include the time taken to render the image.

Below, we present the average evaluation results based on 100 runs of randomized test cases with seed run from 0 to 99

	Greedy	Column Generation	Lazy Initialization
Area Filled Ratio	0.205	0.132	0.187
Filled Ratio	0.203	0.127	0.182
Trim Loss	0.309	0.109	0.213
Time	17.563	32.020	9.248

5.2 Case Study

5.2.1 Case Study 1

Scenario

Factory **X** manufactures a wide array of plastic components and employs five different stock types to cater to various product dimensions.

Available Stock

Table 1: *Available Stock for Factory X*

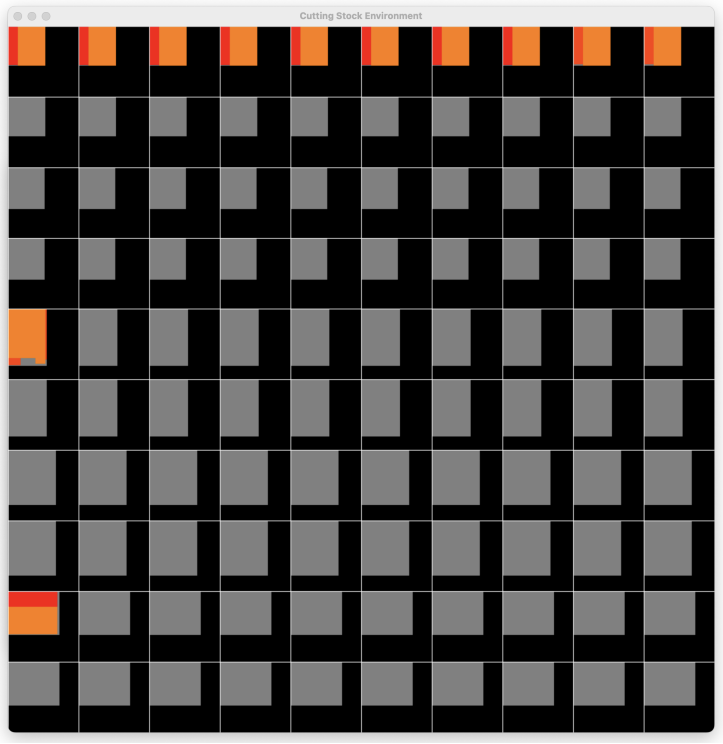
Stock Type	Quantity	Width (cm)	Length (cm)
1	20	53	56
2	20	52	59
3	20	55	81
4	20	68	78
5	20	73	62

Product Requirements

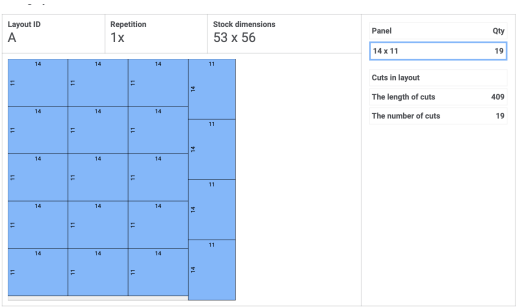
Table 2: *Product Requirements for Factory X*

Product Size (cm)	Quantity Required
11×14	50
18×2	51
39×14	52

Evaluation

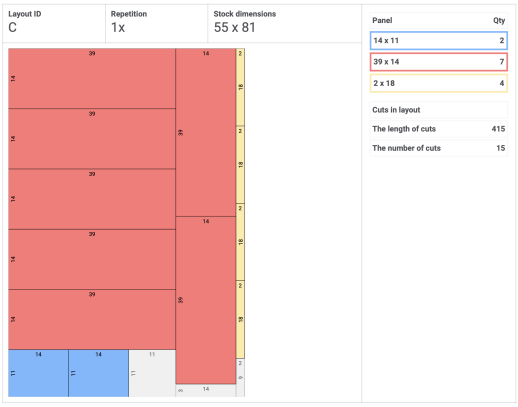


Our Column Generation Output

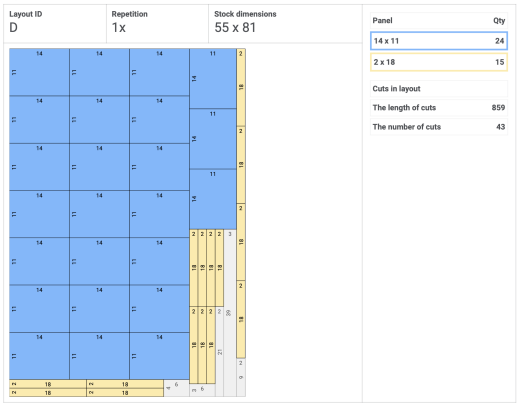


(a) *Figure 1 (optiCutter)*

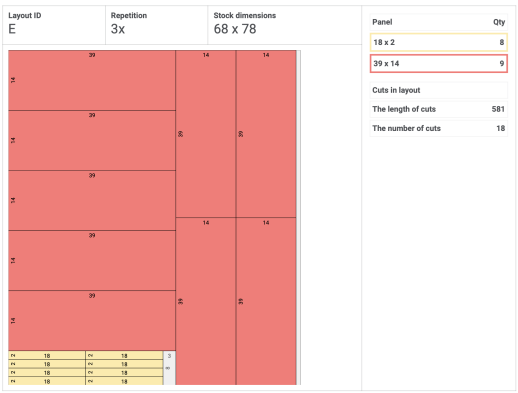
(b) *Figure 2 (optiCutter)*



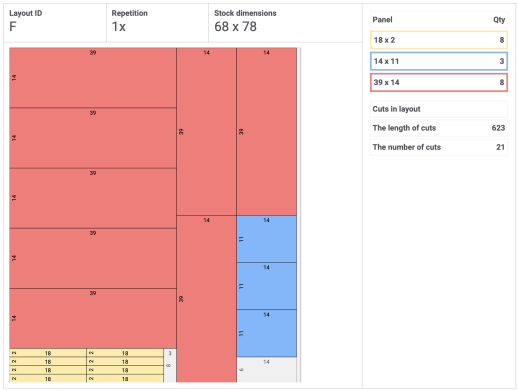
(c) *Figure 3 (optiCutter)*



(d) *Figure 4 (optiCutter)*



(e) *Figure 5 (optiCutter)*



(f) *Figure 6 (optiCutter)*

Table 3: *Evaluation Using Our Code*

	optiCutter	CutList Optimizer	Greedy	Column Generation	Lazy Initialization
Stock Area Used	39030	37928	50456	38661	41552
Area Filled Ratio	0.096	0.093	0.124	0.095	0.102
Filled Ratio	0.090	0.090	0.170	0.120	0.140
Trim Loss	0.028	0.029	0.248	0.015	0.087
Time	-	-	6.510	0.109	18.337

5.2.2 Case Study 2

Scenario

Factory **Y** manufactures a wide array of plastic components and employs five different stock types to cater to various product dimensions.

Available Stock

Table 4: *Available Stock for Factory Y*

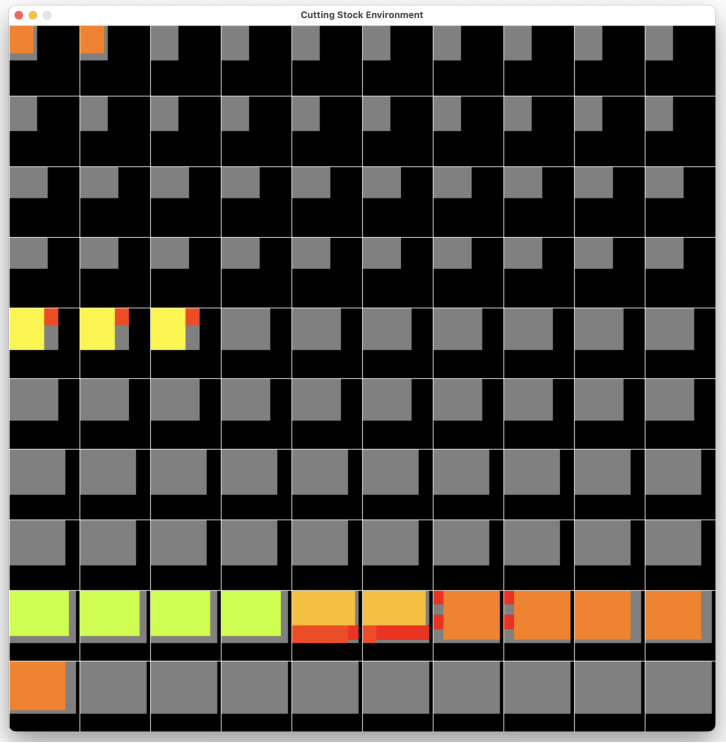
Stock Type	Quantity	Width (cm)	Length (cm)
1	20	40	50
2	20	55	45
3	20	70	60
4	20	80	65
5	20	95	75

Product Requirements

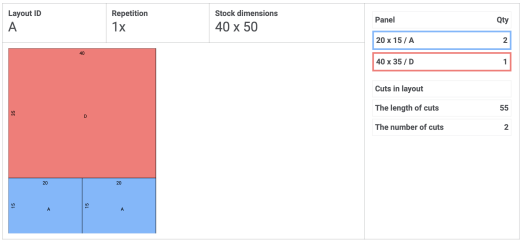
Table 5: *Product Requirements for Factory Y*

Product Size (cm)	Quantity Required
20×15	10
25×20	8
40×35	22
50×45	4
60×50	3
85×65	4

Evaluation

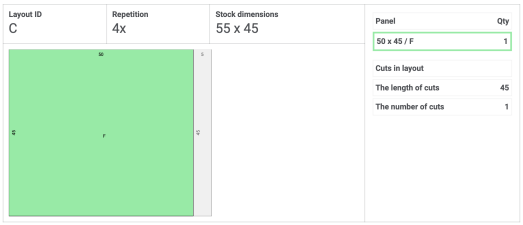


Our Column Generation Output

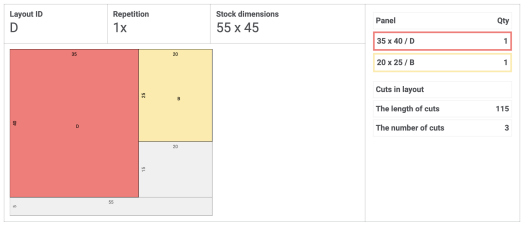


(a) *Figure 1 (optiCutter)*

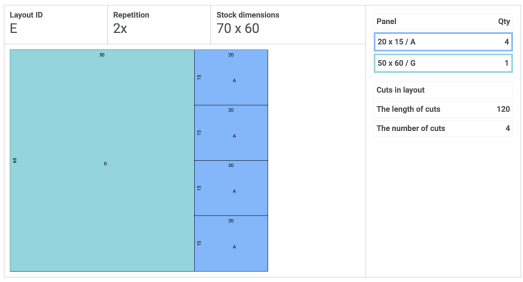
(b) *Figure 2 (optiCutter)*



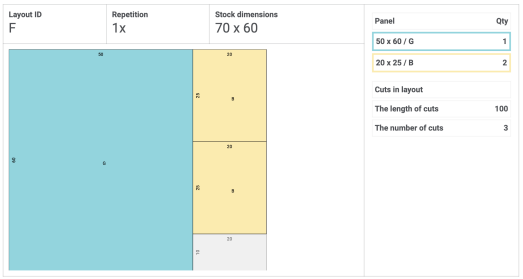
(c) *Figure 3 (optiCutter)*



(d) *Figure 4 (optiCutter)*



(a) *Figure 5 (optiCutter)*

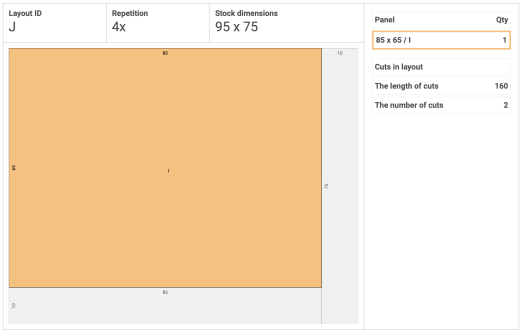


(b) *Figure 6 (optiCutter)*

(c) *Figure 7 (optiCutter)*

(d) *Figure 8 (optiCutter)*

(a) *Figure 9 (optiCutter)*



(b) *Figure 10 (optiCutter)*

Table 6: *Evaluation Using Our Code*

	optiCutter	CutList Optimizer	Greedy	Column Generation	Lazy Initialization
Stock Used Area	91975	95950	110800	94975	100900
Area Filled Ratio	0.219	0.228	0.264	0.226	0.240
Filled Ratio	0.210	0.330	0.390	0.160	0.350
Trim Loss	0.155	0.188	0.315	0.189	0.224
Time	-	-	0.325	0.088	1.307

5.2.3 Case Study 3

Scenario

Factory **Z** produces a range of metal parts and utilizes three distinct types of stocks to accommodate different product sizes.

Available Stock

Table 7: *Available Stock for Factory Z*

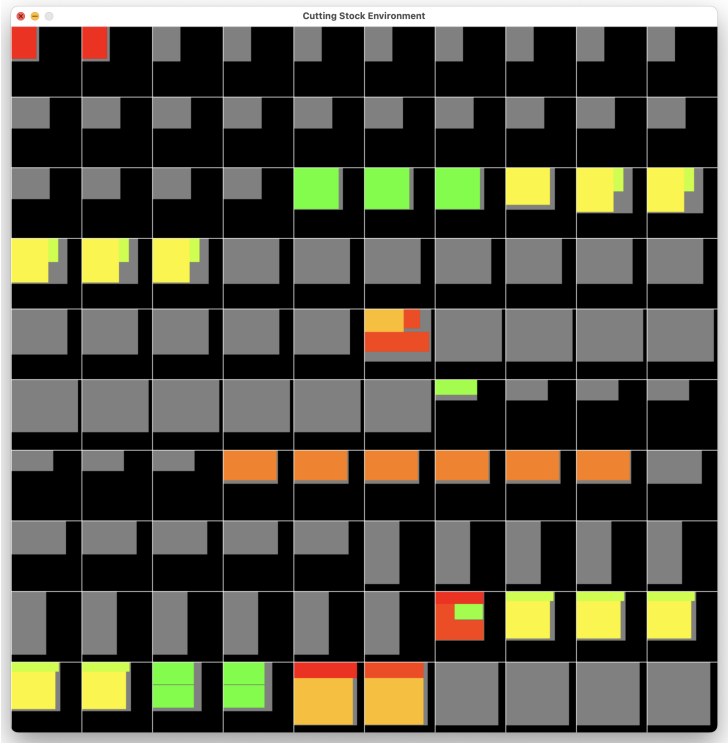
Stock Type	Quantity	Width (cm)	Length (cm)
1	10	40	50
2	14	55	45
3	4	70	60
4	17	80	65
5	11	95	75
6	7	60	30
7	12	78	48
8	11	50	90
9	8	70	70
10	6	90	90

Product Requirements

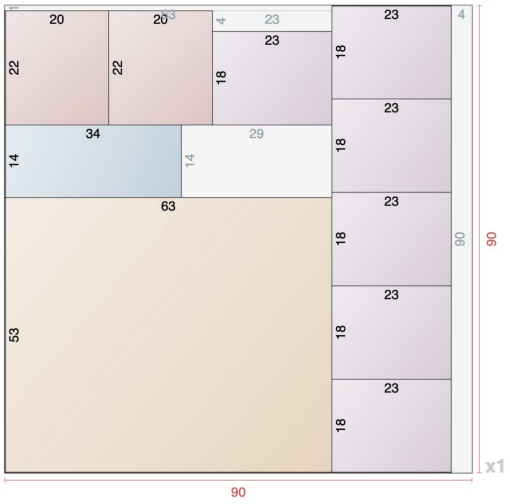
Table 8: *Product Requirements for Z*

Product Size (cm)	Quantity Required
23×18	16
28×23	12
43×38	12
33×28	14
63×53	11
14×34	15
20×22	5
32×59	10

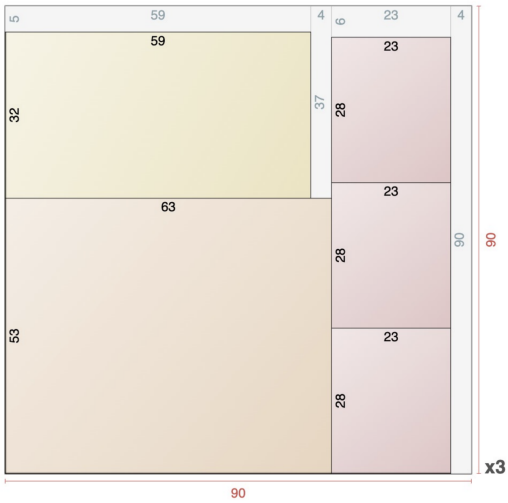
Evaluation



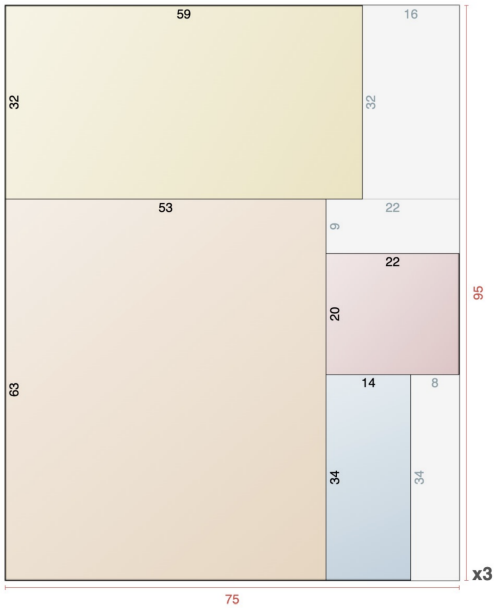
Our Column Generation Output



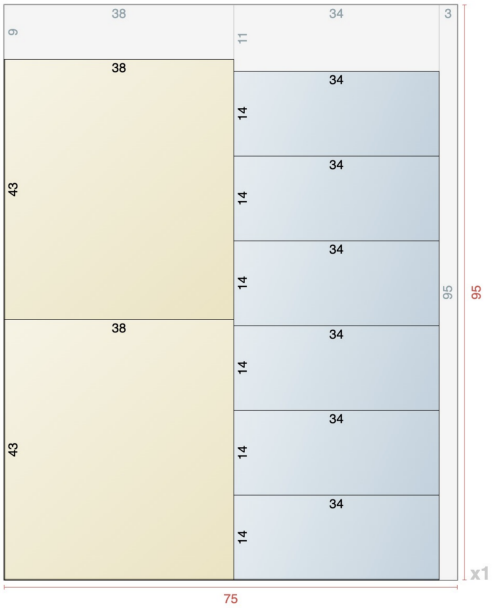
(a) *Figure 1 (CutList Optimizer)*



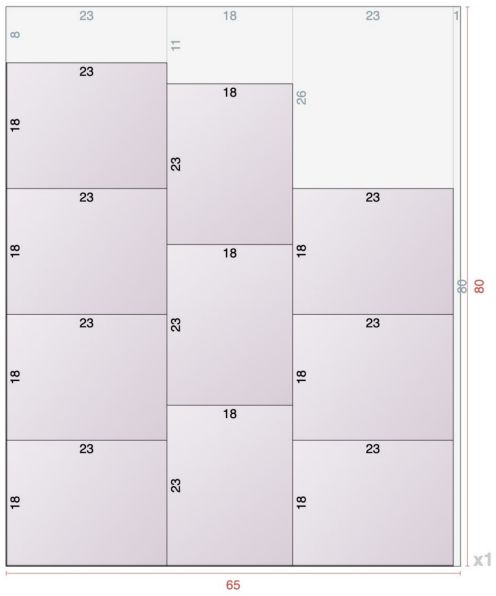
(b) *Figure 2 (CutList Optimizer)*



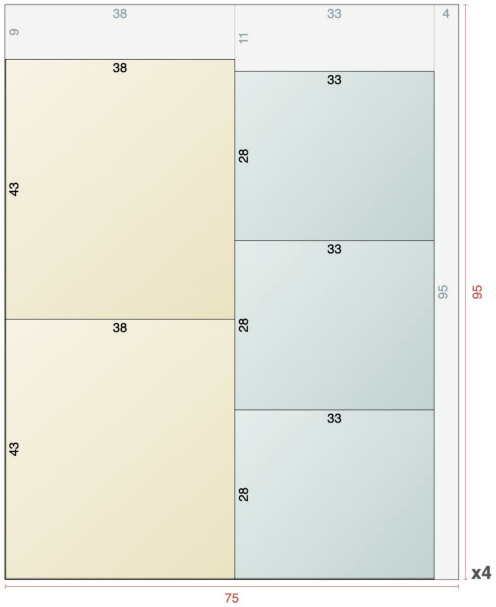
(c) *Figure 3 (CutList Optimizer)*



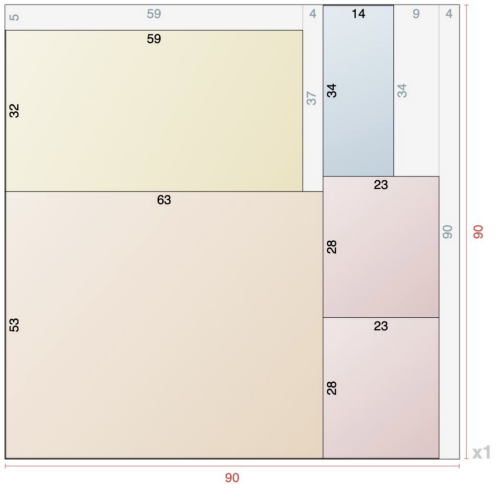
(d) *Figure 4 (CutList Optimizer)*



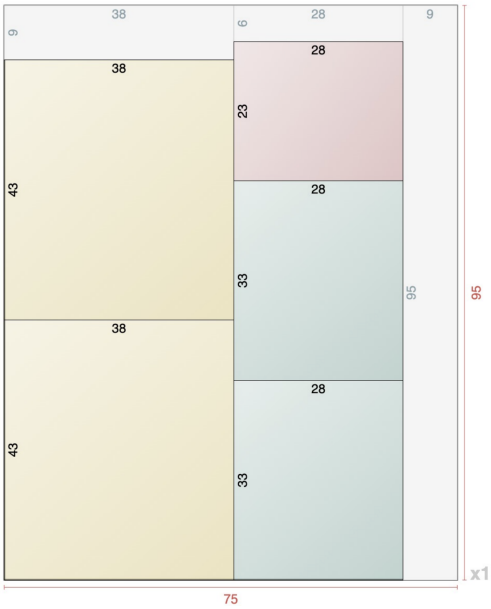
(a) *Figure 5 (CutList Optimizer)*



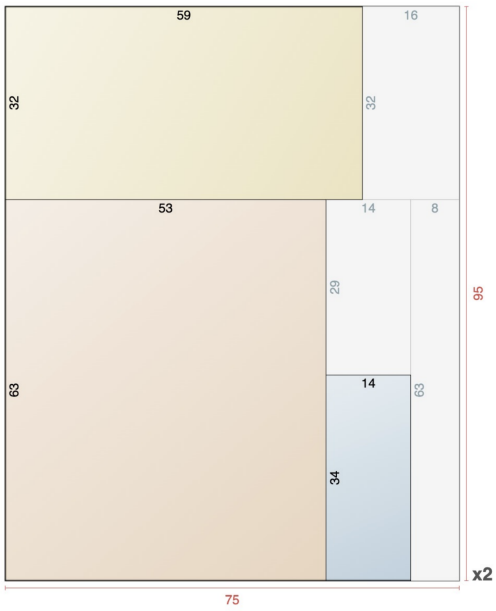
(b) *Figure 6 (CutList Optimizer)*



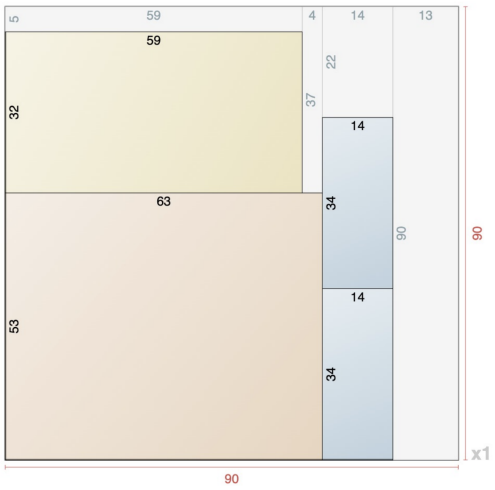
(c) *Figure 7 (CutList Optimizer)*



(d) *Figure 8 (CutList Optimizer)*



(a) *Figure 9 (CutList Optimizer)*



(b) *Figure 10 (CutList Optimizer)*

Table 9: *Evaluation Using Our Code*

	CutList Optimizer	Greedy	Column Generation	Lazy Initialization
Stock Area Used	132175	188350	133589	154650
Area Filled Ratio	0.305	0.435	0.308	0.357
Filled Ratio	0.180	0.490	0.290	0.440
Trim Loss	0.155	0.399	0.164	0.289
Time	-	5.402	1.187	18.787

Summary for Analyze Result

The evaluation results across the three case studies clearly demonstrate that our **Column Generation** algorithm outperforms the **Lazy Initialization** heuristic, achieving superior material utilization and operational efficiency. In each case study, Column Generation consistently delivered better performance compared to Lazy Initialization and Greedy Algorithm, highlighting its effectiveness in optimizing the 2D Cutting Stock Problem. However, in some cases where the amount of stock and item types are large, the column generation needs a significant time to calculate for strip and pattern generation, while lazy initialization doesn't depend on number of types. Furthermore, the performance of Column Generation closely approaches that of established external software solutions such as **optiCutter** and **CutList Optimizer**, underscoring its practical viability and competitiveness in real-world industrial applications. This positions our algorithm as a robust and efficient solution for enhancing operational workflows and reducing material costs in manufacturing environments.

The evaluation results demonstrate that our model effectively optimizes the 2D Cutting Stock Problem, achieving high material utilization and low waste. A key advantage of our algorithm is its implementation of guillotine cuts, which aligns with the practical requirements of businesses and factories. Guillotine cuts simplify the cutting process and are easily automatable in industrial settings, ensuring that our solutions are not only optimal but also feasible for real-world applications. This compatibility with existing manufacturing processes makes our algorithm a valuable tool for enhancing operational efficiency and reducing material costs in various industrial environments.

6 Conclusion

The two-dimensional Cutting Stock Problem (2D CSP) remains a critical challenge in industries aiming to optimize material usage and reduce costs. This report has examined the complexities of 2D CSP and evaluated effective algorithmic approaches, notably the **Lazy Stock Initialization Heuristic** and **Column Generation**.

The Lazy Stock Initialization Heuristic proved advantageous by minimizing initial computational and memory demands through deferred evaluation of cutting patterns. This approach enhances scalability and flexibility, making it suitable for large-scale problems. Meanwhile, Column Generation efficiently manages the exponential number of potential cutting patterns by iteratively incorporating only the most promising ones, thereby improving solution quality and computational efficiency.

Our case studies demonstrated that the proposed models perform competitively against established cutting software, highlighting their practical applicability in real-world scenarios. Despite the benefits, these methods introduce implementation complexities and may incur increased runtime under certain conditions.

Future research could explore hybridizing these techniques with other optimization methods, such as genetic algorithms, to further enhance performance. Additionally, integrating predictive models to identify promising cutting patterns could streamline the optimization process.

In summary, the strategies discussed in this report offer valuable tools for addressing the 2D Cutting Stock Problem, enabling industries to achieve greater material efficiency and cost savings. Continued advancements in optimization algorithms will further bolster the effectiveness and applicability of these solutions in diverse industrial applications.

6.1 Future Research and Extensibility

- **Hybrid Optimization Methods** Combine heuristic algorithms like Genetic Algorithms with exact techniques such as Column Generation to enhance solution quality and reduce computation time.
- **Machine Learning Integration** Utilize machine learning models to predict and prioritize promising cutting patterns, thereby streamlining the optimization process.
- **Real-Time and Dynamic Optimization** Develop algorithms capable of adapting to real-time changes in demand, material availability, and production schedules to improve operational flexibility.
- **Empirical Validation** Conduct extensive empirical studies and benchmark against state-of-the-art methods to validate performance, identify limitations, and guide further improvements through industry collaborations.

7 References

[1] Fabio Furini, Enrico Malaguti, Rosa Medina Durán, Alfredo Persiani, Paolo Toth (2011) *A column generation heuristic for the two-dimensional two-staged guillotine cutting stock problem with multiple stock size*, Operations research

[2] *Lazy Initialization*. Wikipedia. Available at: https://en.wikipedia.org/wiki/Lazy_initialization.

[3] P. C. Gilmore and R. E. Gomory (1961) *A linear programming approach to the cutting-stock problem*, Operations research, vol. 9, no. 6, pp. 849–859.

[4] *First-fit bin packing*. Wikipedia. Available at: https://en.wikipedia.org/wiki/First-fit_bin_packing.

[5] R. W. Haessler and P. E. Sweeney (1991) *Cutting stock problems and solution procedures,*, European Journal of Operational Research, vol. 54, no. 2, pp. 141–150.

[6] M. Iori, V. L. De Lima, S. Martello, F. K. Miyazawa, and M. Monaci (2021) *Exact solution techniques for two-dimensional cutting and packing,*, European Journal of Operational Research, vol. 289, no. 2, pp. 399–415.