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): Nguyễn An Khương

Report team: Group TN01_TN - Team 104
Students: Bùi Ngọc Minh - 2312046 ***(Leader)***
Lê Trọng Thiện - 2313233
Phạm Lê Tiến Đạt - 2310687
Lương Minh Thuận - 2313348
Nguyễn Đăng Hiên - 2310936

HO CHI MINH CITY, DECEMBER 2024

Contents

List of Symbols **3**

List of Acronyms **3**

List of Figures **5**

List of Tables **5**

Member list & Workload **5**

1 Introduction and Motivation **6**
1.1 The Cutting-Stock Problem . . . 6
1.2 Reinforcement Learning . . . 6
1.3 Motivation . . . 6

2 Acknowledge **8**

3 Algorithm **9**
3.1 Heuristic Algorithms . . . 9
3.1.1 Constructive Heuristics . . . 9
3.1.2 First-Fit Algorithm for 2D Cutting-Stock . . . 10
3.1.3 A better solution . . . 11
3.1.4 Combination Algorithm . . . 13
3.1.5 Conclusion . . . 13
3.2 Reinforcement Learning . . . 14
3.2.1 Formal RL model . . . 15
3.2.2 PPO approach to training a RL agent . . . 16
3.2.3 Our group's RL approach to the 2D SCP . . . 17
3.2.4 Conclusion . . . 18
3.3 Using simplex ILP optimization for a heuristic . . . 18

4 Modeling **19**
4.1 Cutting-Stock Problem . . . 19
4.1.1 Problem statements . . . 19
4.1.2 Formalization . . . 19
4.2 Column generation and the 2D knapsack problem . . . 22
4.2.1 2 stages guillotine (with trimming) . . . 23
4.2.2 general 3 stages guillotine . . . 26
4.2.3 Extending the formulation to higher dimensions . . . 27

5 Case study **28**
5.1 Problem Description . . . 28
5.2 Optimization Problem . . . 30
5.3 Solution . . . 31
5.3.1 Our algorithms . . . 31
5.3.2 Online tool . . . 33

6 Evaluate the algorithms **35**
6.1 Frameworks 35
6.2 Results after execution 35
6.3 Evaluation 37

7 Conclusion **38**
7.1 Source code, data and related materials 38
7.2 Afterword 38

References **39**

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

ILP Integer Linear Programing problem

RL Reinforcement learning

s.t. Subject to (in context of ILP)

PPO Proximal policy optimization (in context of RL)

CNN Convolutional neural network

List of Figures

1 Visualizing the RL framing . . . 15
2 Visualizing orthogonal space $\boldsymbol{R}$ . . . 20
3 Example formulation of $\boldsymbol{E(1,\ 4,\ 1,\ 3)}$ . . . 21
4 Visualization of a strip . . . 24
5 Visualization of the stacking of strips . . . 25
6 A situation where a stage 3 cut can split a "column" . . . 26
7 The raw materials – marble blocks . . . 28
8 The two dimensional panels . . . 29
9 The final products . . . 30
10 Combination heuristic cutting patterns for this case study . . . 31
11 RL agent's produced cutting patterns for this case study . . . 31
12 Input to optiCutter - List of available Stock panels . . . 33
13 Input to optiCutter - List of Ordered items . . . 33
14 Opticutter's statistics on the its solution to this case study . . . 34
15 Comparison of Average Waste Rates Across Algorithms . . . 37
16 Comparison of Average Fitness Across Algorithms . . . 37
17 Comparison of Average Time Across Algorithms . . . 37

List of Tables

1 Member list & workload . . . 5
2 Comparison between First-Fit and Best-Fit Algorithms . . . 12
3 Comparison of First-Fit, Best-Fit, and Combination Algorithms . . . 14
4 Case study - Customer's Order . . . 30
5 Case study - Available panels' Data . . . 31
6 Item to color mapping used to display cutting patterns . . . 32
7 Greedy Algorithm (First fit) bench mark results . . . 36
8 Combination Heuristic Algorithm bench mark results . . . 36
9 Reinforcement Learning agent bench mark results . . . 36

Member list & Workload

No.	Full name	ID	Problems	% done
1	Bùi Ngọc Minh	2312046	- Formulation of the 2D CSP as an ILP, including the restricted 2D knapsack problem - Algorithm research (Theory behind RL and PPO) - LaTeX	100%
2	Lê Trọng Thiện	2313233	- Algorithm research & implementation (Combination heuristic derived from best fit and first fit algorithm)	100%
3	Phạm Lê Tiến Đạt	2310687	- Algorithm research & implementation (Simplex optimization on a predefined set of cutting patterns heuristic)	100%
4	Lương Minh Thuận	2313348	- Algorithm research & implementation (RL, training via PPO) - Collect benchmark data on the different policies	100%
5	Nguyễn Đăng Hiên	2310936	- Algorithm research (first fit and best fit heuristics) - Case study of a real life instance of the 2D CSP (Real life data collection and problem formulation) - LaTeX	100%

Table 1: Member list & workload

1 Introduction and Motivation

1.1 The Cutting-Stock Problem

The Cutting-Stock Problem is a well-known NP-hard optimization problem with significant real-world implications. It can be stated as follows:

Given a set of stock materials with fixed dimensions and a list of items to be cut from these stocks with specified dimensions and demands, determine a cutting pattern that minimizes the total number of stocks used or the amount of waste generated. The goal is to meet all demands while optimizing resource utilization.

This problem frequently arises in industries such as manufacturing, logistics, and supply chain management, where raw materials such as sheets, rods, or rolls need to be cut into smaller pieces efficiently.

The Cutting-Stock Problem is typically formulated as an Integer Linear Programming (ILP) problem. Various techniques have been proposed to solve it, including branch-and-bound algorithms, heuristic methods, and column generation approaches with trade off between computational complexity and solution quality between them.

For simplicity, this big assignment only focuses on a specific variation of the problem, namely the a variation of the 2D Cutting-Stock Problem with each items being non-rotatable 2D rectangles whose dimensions are all integers.

1.2 Reinforcement Learning

Reinforcement Learning (RL) is a subfield of machine learning that focuses on training an agent to interact with a dynamic environment. The agent learns to take actions that maximize its cumulative reward through trial-and-error interactions. Unlike supervised learning, RL does not require explicit labeling of data or immediate corrections for incorrect behavior. Instead, it relies on feedback in the form of rewards or penalties to guide the agent toward optimal behavior.

One advantage of RL is its ability to adapt to complex, high-dimensional problems where traditional methods struggle. In the context of the Cutting-Stock Problem, RL offers the potential to dynamically generate cutting patterns by learning from past performance, which could lead to new innovative and efficient solutions.

1.3 Motivation

Our group's objectives in this project are to:

- **1)** Formalize the Cutting-Stock Problem as an ILP, providing a clear mathematical representation of the problem constraints and objectives.
- **2)** Analyze, discuss, and implement different approaches to solving the problem, including traditional optimization techniques and a Reinforcement Learning-based approach. These perspective allows us to compare the strengths and weaknesses of both methods in tackling this challenging problem.
- **3)** Propose potential extensions to our model and exploring its applications in real life situations and demands.

By combining insights from different approaches, this project aims to develop a comprehensive understanding of the Cutting-Stock Problem and its potential solutions. We hope that our work will contribute to the broader field of Operations Research and inspire future applications of machine learning in industrial problems.

2 Acknowledge

Our group expresses deep gratitude to our instructor, Mr. Nguyễn An Khương for his guidance and valuable feedback throughout the creation process of this big assignment.

We would also like to thank the whole of HCMUT community of students and lecturers, whose questions and answers on the forum helped push the project through hard decisions and ambiguity.

3 Algorithm

3.1 Heuristic Algorithms

Heuristic algorithms are problem-solving methods that focus on finding approximate solutions efficiently, especially in scenarios where exact solutions are computationally expensive or infeasible. These methods rely on rules of thumb, domain knowledge, or practical strategies to reduce the complexity of the search space. By strategically ignoring less promising solutions, heuristics aim to provide good, though not necessarily optimal, results within a reasonable time frame.

In the context of the Cutting-Stock Problem, heuristic algorithms are instrumental in generating feasible cutting patterns while balancing computational effort and solution quality. This is achieved by incorporating constraints, such as limiting the number of cutting stages or enforcing specific cutting styles like guillotine cuts. These constraints significantly simplify the problem by reducing the vast space of potential cutting patterns, allowing for practical solutions that meet demand and minimize material waste. Despite their approximate nature, heuristics are often highly effective in real-world industrial applications, where quick decisions and resource optimization are paramount.

Heuristic methods are widely applied in solving two-dimensional cutting and packing problems due to their ability to provide high-quality solutions within a reasonable computational time. These methods are particularly relevant to the Cutting-Stock Problem, as they offer efficient ways to handle its NP-hard nature without requiring an exhaustive exploration of the solution space.

3.1.1 Constructive Heuristics

Constructive heuristics build solutions incrementally by placing items into stock one at a time. Three classical strategies that have laid the foundation for many solution methods are:

- **First-Fit Algorithm:** This method places each item in the first available space or bin that can accommodate it without violating constraints. It is simple, computationally efficient, and often used as a baseline for comparison in packing problems. While not guaranteed to produce optimal solutions, it provides a quick and feasible layout that can be further refined using advanced techniques.
- **Bottom-Left Algorithm:** This method places items sequentially in the lowest possible position, left-justified. It is effective for structured packing problems, where straightforward placement rules lead to reasonably efficient layouts with minimal computational effort.
- **Best-Fit Approach:** This method selects items that best fit into the lowest available space during each placement. By focusing on optimal utilization of stock material, it enhances efficiency and reduces waste, making it particularly suitable for resource-constrained problems.

Efficient implementations of these algorithms have been extensively studied. For instance, Chazelle and Imahori & Yagiura proposed optimized versions to improve computational performance while maintaining solution quality. These constructive heuristics serve as the building blocks for more sophisticated methods and are often used to generate initial solutions in hybrid algorithms.

As seen in the source code provided in the description of this assignment, the approach of the source code is primarily designed around the First-Fit method, which will be discussed in more detail in the next section of the report.

3.1.2 First-Fit Algorithm for 2D Cutting-Stock

The **First-Fit Algorithm** is a straightforward heuristic used to solve 2D cutting-stock problems, where the goal is to allocate rectangular items (or products) onto stock sheets in a way that minimizes waste. The algorithm prioritizes placing each product on the first stock sheet that can accommodate it, balancing simplicity and efficiency. Although the solution is not guaranteed to be optimal, it is computationally efficient and feasible for many practical scenarios.

The algorithm operates as follows:

1. **Product Selection:** Products are placed one by one, in the order they are provided.
2. **Find the First Suitable Stock:** For each product, the algorithm checks each available stock sheet (in the order they are considered) to determine if the product can fit, based on the stock's remaining available area and dimensions.
3. **Place the Product:** If the product can fit on an existing stock sheet, it is placed there. If no existing stock sheet can accommodate the product, a new stock sheet is introduced, and the product is placed on it.
4. **Continue:** The process repeats until all products have been placed.

Let:

- n be the number of products to cut.
- m be the number of stock sheets used.
- W_i, H_i be the width and height of the i-th product.
- $W_j^{\text{stock}}, H_j^{\text{stock}}$ be the width and height of the j-th stock sheet.

The objective of the First-Fit Algorithm for 2D Cutting-Stock is to minimize the number of stock sheets required, while ensuring that the placement respects the dimensions and capacities of the stock sheets. The placement process can be formalized as:

1. Find the first stock sheet j such that ($W_i \leq$ remaining width) and ($H_i \leq$ remaining height).
2. If no existing stock sheet can accommodate the product, a new stock sheet is created, then $m = m + 1$ (introduce a new stock sheet).

This approach can be enhanced with additional strategies for handling rotation of products, prioritizing smaller items, or rearranging the placement order to achieve better results.

The main idea is to cut from the largest stock to the smallest stock, and from the largest product to the smallest product, in such a way that the area cut from the stock changes as little as possible. Therefore, the following pseudo code outlines the First-Fit Algorithm. The time complexity of the First-Fit algorithm for 2D Cutting-Stock is $O(n \cdot m)$, where:

- n is the number of products to be placed.
- m is the number of stock sheets used.

Function First_fit_2DCSP:
  push one of each stock_type into the stock_list.
  for $stock_i$ **from** the largest **to** smallest area in stock_list:
    for $prod_j$ **from** the largest **to** smallest area in product_list:
      if $prod_j$ **can fit** into $stock_i$:
        place $prod_j$ **into** $stock_i$
      else:
        let "stockType" **be** the first stock_type that **fits** $prod_j$
        create a new stock of type "stockType"
        place $prod_j$ **into** the new stock
      if the textbfdemand for $prod_j$ is textbfmet:
        remove $prod_j$ **from** the item list

Each product might require checking all existing stock sheets to find a place, and if no stock sheet is suitable, a new stock sheet is introduced.

The First-Fit Algorithm is a simple and effective heuristic for solving 2D cutting-stock problems. While it does not guarantee an optimal solution, it performs well in many practical scenarios due to its ease of implementation and adaptability to various requirements. However, the algorithm has limitations in terms of material utilization efficiency and cutting complexity, which leaves room for improvement.

In this report, we aim to explore a better solution, focusing on developing a heuristic algorithm that strikes a balance between reducing cutting complexity and maximizing material utilization. The following sections will delve into this improved algorithm and its potential to outperform the First-Fit approach in key performance metrics.

3.1.3 A better solution

As mentioned in the previous section, the First Fit algorithm provided in the sample code is suitable for large-scale problems due to its fast execution speed, but it has limited optimization capabilities. On the other hand, we have another algorithm called Best Fit, which is better at minimizing material waste but comes at the cost of higher computational effort.

Introduction to the Best Fit Algorithm

The **Best-Fit Algorithm** is a heuristic method commonly used to solve 2D cutting-stock problems, aiming to minimize material waste when allocating rectangular items (or products) onto stock sheets. Unlike the First-Fit algorithm, Best-Fit attempts to place each product on the stock sheet that leaves the least remaining unused area, leading to better material utilization.

The Best-Fit Algorithm operates as follows:

1. **Product Selection:** Products are placed one by one, following the order they are given.
2. **Find the Best Suitable Stock:** For each product, the algorithm examines each available stock sheet (in the order considered) to determine which one leaves the smallest remaining area after placing the product. This ensures that the least amount of unused space remains on the stock sheet.

3. **Place the Product:** If a product fits on an existing stock sheet, it is placed in the position that leaves the smallest leftover area. If no existing stock sheet can accommodate the product, a new stock sheet is introduced, and the product is placed on it.
4. **Continue:** This process is repeated until all products have been placed.

Let:

- n be the number of products to cut.
- m be the number of stock sheets used.
- W_i, H_i be the width and height of the i-th product.
- W_{stock_j}, H_{stock_j} be the width and height of the j-th stock sheet.

The objective of the Best-Fit Algorithm for 2D Cutting-Stock is to minimize the unused area on stock sheets, thereby reducing material waste. The placement process can be formalized as follows:

- For each product, find the stock sheet j such that the remaining area after placing the product is minimized:

  $$\text{minimize remaining area where } W_i \leq \text{remaining width and } H_i \leq \text{remaining height.}$$

- If no existing stock sheet can accommodate the product, a new stock sheet is created:

  $$m = m + 1 \quad \text{(introduce a new stock sheet).}$$

Approaching a better solution

Compared to the First-Fit algorithm, we can draw the following brief conclusion in the table:

Aspect	First-Fit Algorithm	Best-Fit Algorithm
Approach	Place each product on the first stock sheet that can accommodate it	Place each product on the stock sheet that leaves the least unused space
Efficiency	Computationally faster	More computationally expensive due to evaluating all stock sheets
Material Waste	Generally higher material waste	Typically reduces material waste by better fitting products
Optimality	Does not guarantee optimal solutions	Provides better solutions in terms of material utilization, but still not optimal
Use Case	Suitable for large problems where speed is a priority	Suitable for cases where minimizing material waste is more important than speed

Table 2: Comparison between First-Fit and Best-Fit Algorithms

Since each algorithm has its own strengths, we propose an algorithm that optimizes both execution time and result quality. This is a combination of the two algorithms above, which we call the "Combination Algorithm", and it will be presented in the following section.

3.1.4 Combination Algorithm

The Combination Algorithm is a novel heuristic designed to optimize both execution time and material utilization in the 2D cutting-stock problem. By combining the strengths of the First-Fit and Best-Fit algorithms, this approach seeks to strike a balance between computational efficiency and minimizing material waste.

While the First-Fit algorithm excels in speed and simplicity, it may leave more unused space on the stock sheets, leading to higher material wastage. On the other hand, the Best-Fit algorithm, while better at reducing material waste, requires more computational resources and may be slower in execution. The Combination Algorithm leverages the strengths of both approaches to optimize the cutting process: it first uses the First-Fit algorithm to quickly place products on stock sheets and then applies the Best-Fit algorithm to refine the placement for better material utilization.

The algorithm operates as follows:

1. **Initial Placement:** Products are placed on stock sheets using the First-Fit algorithm, where each product is placed on the first available stock sheet that can accommodate it.
2. **Refinement:** After the initial placement, the Best-Fit algorithm is applied to adjust the positions of the products, aiming to minimize the remaining unused space on the stock sheets.
3. **Continue:** The process repeats until all products are placed, with the combination of fast placement and optimal adjustment ensuring both efficiency and reduced material waste.

The objective of the Combination Algorithm is to reduce material wastage while maintaining a reasonable execution time, making it an ideal solution for both large-scale and material-efficient cutting problems. Here a pseudo code outlines the Combination Algorithm:

Function Combination_2DCSP:
  for $stock_i$ **from** the largest **to** smallest area in stock list:
    for $prod_j$ **from** the largest **to** smallest area in product list:
      for (x, y) **in** all position can cut $prod_j$ **from** $stock_i$:
        if cut $prod_j$ at position (x, y):
          let S_{ij} be the area of the smallest rectangle contain cut area
      let (x_0, y_0) be the position **where** the value of S_{ij} is smallest
      cut the $prod_j$ **from** $stock_i$ **at** position (x_0, y_0)
  for $stock_i$ **from** the smallest **to** largest area in cut stock list:
    for $stock_j$ **from** the smallest **to** largest area in uncut stock list:
      if cut area of $stock_i$ is smaller than $stock_j$ **then**:
        move cut set of $stock_i$ **to** $stock_j$

3.1.5 Conclusion

The Combination Algorithm is designed to optimize both execution time and result quality by combining the strengths of the First-Fit and Best-Fit algorithms. By leveraging the speed of the First-Fit approach and the material waste reduction capability of the Best-Fit approach, this hybrid algorithm aims to deliver efficient solutions with improved resource utilization. The Combination Algorithm works by first applying the First-Fit method to quickly allocate products

and then refining the placements using the Best-Fit approach to minimize the remaining unused space.

This approach strikes a balance between computational efficiency and optimality, offering a practical solution for large-scale cutting-stock problems where both time and material waste are critical factors. Compared to the individual algorithms, the Combination Algorithm provides a more optimized solution by addressing both aspects in a complementary manner.

Aspect	**First-Fit Algorithm**	**Best-Fit Algorithm**	**Combination Algorithm**
Approach	Place each product on the first stock sheet that can accommodate it	Place each product on the stock sheet that leaves the least unused space	Hybrid approach combining First-Fit for quick allocation and Best-Fit for optimization
Efficiency	Computationally faster	More computationally expensive	Balanced, faster than Best-Fit but more optimal than First-Fit
Material Waste	Generally higher material waste	Typically reduces material waste by better fitting products	Reduced material waste due to the optimization step after First-Fit
Optimality	Does not guarantee optimal solutions	Provides better solutions in terms of material utilization, but still not optimal	More optimized than both First-Fit and Best-Fit by balancing speed and waste reduction
Use Case	Suitable for large problems where speed is a priority	Suitable for cases where minimizing material waste is more important than speed	Ideal for large-scale problems where both speed and waste reduction are important

Table 3: Comparison of First-Fit, Best-Fit, and Combination Algorithms

3.2 Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning designed to train an agent to interact with a dynamic environment by learning from trial-and-error experiences. The agent seeks to maximize cumulative rewards over time, making decisions based on feedback received from the environment in the form of rewards or penalties. Unlike supervised learning, RL does not require explicitly labeled data or immediate feedback for every action. Instead, it relies on delayed rewards, allowing the agent to evaluate the long-term effects of its actions.

For the Cutting-Stock Problem, RL presents a promising approach to dynamically generating cutting patterns that adapt over time. By leveraging past performance, an RL agent can explore innovative and efficient solutions that might not be apparent through traditional optimization techniques. For example, the agent could learn to balance competing objectives such as minimizing material waste while maintaining flexibility to fulfill diverse demands. Moreover, RL's ability to handle high-dimensional and complex problems makes it a suitable candidate for addressing variations of the Cutting-Stock Problem that involve multiple constraints or dynamic requirements.

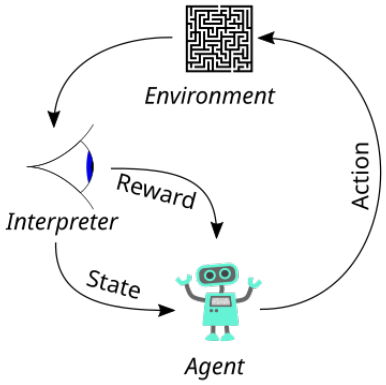

Figure 1: Visualizing the RL framing

3.2.1 Formal RL model

RL is often modeled as a Markov decision process as follows:

- A set of environment and agent states $\boldsymbol{S}$, i.e the state space.
- A set of actions of the agent $\boldsymbol{A}$, i.e the action space
- Given any two state $\boldsymbol{s}$, $\boldsymbol{s^*}$ and an action $\boldsymbol{a} \in \boldsymbol{A}$, a probability that represents the likelihood that taking action $\boldsymbol{a}$ in state $\boldsymbol{s}$ will make the environment transition to state $\boldsymbol{s^*}$:

  $$\boldsymbol{P_a(s, s^*) = Probability(S_{t+1} = s^* | S_t = s, A_t = a)}$$

  Where $\boldsymbol{S_t}$ is the state at time $\boldsymbol{t}$
- The immediate reward after transitioning from state $\boldsymbol{s}$ to state $\boldsymbol{s^*}$ under action $\boldsymbol{a}$, denoted by $\boldsymbol{R_a(s, s^*)}$.

In a typical RL framing, the agent interacts with the environment in discrete steps. At any time step $\boldsymbol{t}$, the agent receives the current state $\boldsymbol{S_t}$ and reward at that time step $\boldsymbol{R_t}$. The agent then picks an action $\boldsymbol{A_t} \in \boldsymbol{A}$ to send to the environment. The environment then moves to a new state $\boldsymbol{S_{t+1}}$ with a new reward $\boldsymbol{A_{t+1}}$ based on $\boldsymbol{P_{A_t}}$.

The agent picks action based on a policy, which is defined as a probability distribution on $\boldsymbol{S} \times \boldsymbol{A}$, denoted by $\boldsymbol{\pi}$:

$$\boldsymbol{\pi : S \times A \to [0, 1]}$$

The policy represents the probability of selecting an action $\boldsymbol{a}$ under a given state $\boldsymbol{s}$:

$$\boldsymbol{\pi(s, a) = Probability(A_t = a | S_t = s)}$$

The estimated value of a state is given by a state-value function, defined as the expected discounted return starting with state $\boldsymbol{s}$, i.e starting with state $\boldsymbol{S_0 = s}$ and successively follows policy $\boldsymbol{\pi}$:

$$\boldsymbol{V_\pi(s) = E(G | S_0 = s)}$$

Where $\boldsymbol{G}$ denotes the discounted return and is defined as the cumulative sum of future discounted rewards:

$$\boldsymbol{G = \sum_{t=0}^{\infty} \gamma^t \cdot R_{t+1} = R_1 + \gamma \cdot R_2 + \gamma^2 \cdot R_3 + ...}$$

Where $\boldsymbol{R_{t+1}}$ denotes the reward for the transition from state $\boldsymbol{S_t}$ to state $\boldsymbol{S_{t+1}}$.
The calculation of $\boldsymbol{G}$ uses the discount rate $\boldsymbol{\gamma}$, which is a constant that lies between 0 and 1, ($\boldsymbol{\gamma \in [0, 1)}$). The purpose is to prioritize immediate rewards rather than future rewards by weighing the immediate rewards more.

3.2.2 PPO approach to training a RL agent

There have been many method proposed to train a RL agent, including Monte-Carlo methods, policy gradient methods, function approximation methods, etc. For this big exercise, our group focuses on a on-policy, model-free approach called proximal policy optimization (PPO). PPO was developed by John Schulman in 2017. PPO improves upon trust region policy optimization (TRPO) by removing complicated trust region constraints and using a clipping function instead, leading to faster computation. PPO has proven success in many complex tasks including controlling robotic arms or playing online game (namely the DOTA 2).

In PPO, agents can freely explores the environment by executing a series of consecutive actions (an episode), only after the completion of the episode does the policy gets updated. An agent's policy is represented by a neural network, where $\boldsymbol{\theta}$ denotes the policy network's parameters. The training methods necessitates the training of another neural network (the critic), with the goal is to estimate the expected discounted sum of rewards after an episode. PPO also introduces a few functions to train the two neural networks under policy gradient laws:

- $\boldsymbol{A}$ denotes the advantage function. The advantage function tries to estimate the value of a given action. It is calculated as:

  $$\boldsymbol{A = Q - V}$$

  Where $\boldsymbol{Q}$ denotes the discounted sum of rewards after an episode and $\boldsymbol{V}$ denotes the estimated value from the critic. if $\boldsymbol{A > 0}$, the actual return is better than the return from past experiences while $\boldsymbol{A < 0}$ means actual return is worse.

- $\boldsymbol{r_t(\theta)}$ for an action $\boldsymbol{a}$ in state $\boldsymbol{s}$ denotes the ratio function, calculated by:

  $$\boldsymbol{r_t(\theta) = \frac{Probabity(A_t = a, S_t = s)\ under\ \theta}{Probability_{\theta^*}(A_t = a, S_t = s)\ under\ \theta^*}}$$

  Where $\boldsymbol{\theta^*}$ denotes the policy network parameters of the previous policy.
  If $\boldsymbol{r_t(\theta) > 1}$ then action $\boldsymbol{a}$ in state $\boldsymbol{s}$ is more prioritized in the current policy than the previous policy. While if $\boldsymbol{r_t(\theta) \in [0, 1]}$ then action $\boldsymbol{a}$ in state $\boldsymbol{s}$ is less prioritized in the current policy than the previous policy.

- $\boldsymbol{E}$ denotes the objective function of PPO, which is the function the agent aims to optimizes. The function is calculated by:

  $$\boldsymbol{E = min(r_t(\theta) \cdot A\ ,\ clip(r_t(\theta)) \cdot A)}$$

  Where the clip function clipped the input to the range $\boldsymbol{[1 - \epsilon, 1 + \epsilon]}$ This clipping behavior penalizes policy updates that are too big. And by taking the min, the agent is making the safest possible update to its policy by only updating to the smaller possible known bound.

This method has the advantage of being able to optimizes further using gradient descend, making for faster calculations compared to PPO's predecessor TRPO.

3.2.3 Our group's RL approach to the 2D SCP

Due to PPO's quick adaptation and proven success in providing interesting solution to complicated problem. Our group implemented an approach using RL trained using PPO to the 2D SCP.

For a full model, the network has 4 decision variables as outputs:

- The item it decides to cut
- The stock it chooses to cut the item
- The coordinates of the item on the stock ($[\boldsymbol{x}, \boldsymbol{y}]$)

Due to the need to adapt to different environments (different item set and stock set). Our group opted to use 3 CNN (convolutional neural networks) as a policy network instead of linear networks. Two of which handles the filtering of data, which aims to reduce noise and irrelevant information which may confuses the network.

Yet, during the process of training our agent, the network gets really slow when tasked with choosing all 4 variables. Our group decided for the network to only selects the position to cut a particular item on a particular stock. While the decision of which item to cut and which stock to cut the item is delegated to a heuristic approach.

The specific procedure of the algorithm is as follows:

- Chooses an item and a stock to cut based on the above heuristic.
- Forward the current state of the cutting process to the agent, along with the item and stock to cut.

  With a state includes all available stocks along with the items already cut on those stocks and their positions.
- Cut the item into the stock, updating the state of the cutting process.
- Repeat until all demands is met.

For training the agent, a custom reward function is used, our group decided on the grading criteria as follows:

- Reduce the reward heavily if there are empty spaces above the item.
- Reduce the reward heavily if there are empty spaces to the left the item.
- Increase the reward slightly based on how low the coordinates of the item ($[\boldsymbol{x}, \boldsymbol{y}]$) is on the stock, with the coordinates begin at $[\boldsymbol{0}, \boldsymbol{0}]$ on the top left corner and increases to $[\boldsymbol{W_s}, \boldsymbol{H_s}]$ on the bottom right corner.

Our goal with these grading criteria is for the network to not leave gaps around the top left border of an item, leading to fewer wasted spaces.

3.2.4 Conclusion

The RL approach of our group has both advantages and disadvantages. The agent performs really fast and scales fairly well with large data sizes. And the solution obtained from the agent is close to the optimal solution in most cases. Yet given our agent do not select the item and the stock, the approach fails to optimizes the result beyond a heuristic approach like first fit or best fit.

3.3 Using simplex ILP optimization for a heuristic

An optimal approach to solving the 2D SCP is to optimize the problem from the lens of ILP, following the formulation in 4.2. The problem with the aforementioned generalized formulation is that it requires knowing the set $\boldsymbol{J}$ containing all valid cutting patterns, which is hard to systematically without imposing constraints on valid cutting patterns.

One possible heuristic the group proposes here work under this formulation, but done on a restricted set of cutting patterns $\hat{\boldsymbol{J}}$. This set is initially generated by placing random products onto random stocks using another heuristic, like best fit, first fit or our combination algorithm above. (see 3.1.4). This can produces a fairly optimized set of cutting patterns if the set $\hat{\boldsymbol{J}}$ is sufficiently big.

The specific steps of this heuristic algorithm is as follows:

- Generate set $\hat{\boldsymbol{J}}$ using another heuristic up to a big enough size. (1)
- Perform simplex optimization to choose the correct combination of cutting patterns that have the smallest cost. (2)

To turn this method into a sequence of actions for a policy, our group cached the actions that produces each cutting patterns from step (1). The policy can later reference this cached information to produce a sequence of action.

However, due to time constraints, as well as facing memory difficulties, our group decided not to make this method a policy and only implemented it on its own without caching. Furthermore, in practice, this method do not produces much difference between the normal heuristics used and the optimized result under small data sets (Number of unique elements of $\hat{\boldsymbol{J}} \leqslant \mathbf{1000}$). Perhaps it can make a difference in larger data sets.

4 Modeling

4.1 Cutting-Stock Problem

4.1.1 Problem statements

Given a list of rectangular sheets, i.e stocks, whose widths and heights are all integers and may differ from each other. And a list of smaller rectangular items that we want to cut from such stocks. (whose widths and heights are also in integers). Each item is also associated with a demand, which is the number of such item that we are required to fulfilled. The goal is to produce a sequence of ways to cut the stocks (a sequence of cutting patterns) into such items to meet their respective demands.

Considering the practicality and the scope of the problem, our group have chosen a few limitations to apply to a valid cut as follows:

- **1)** Items are non-rotatable. Note that this limitation however can be assumed without loss of generality for models that allows rotations in multiples of 90 degrees due to the transformability of the non-rotatable model into such model, this fact is elaborate further in the latter section.
- **2)** Feasible cutting patterns must only consist of guillotine cuts (orthogonal bisecting cuts that go from one edge to another, exact specification is given below). This assumption is made with practicality in mind as industrial cutting machines can only make such cuts. Variants of this restriction will be discussed in the section for column generation but our group focused mainly on the formulation for staged guillotined cuts, specifically for 2 and 3 stages.

4.1.2 Formalization

The following formulation is based on the work of P. C. Gilmore and R. E. Gomory in 1965 [6] with some adaptations to our specific instance of the problem.

Let $\boldsymbol{S}$ denotes the set of all available stocks and $\boldsymbol{s}$ denotes a member of $\boldsymbol{S}$. A stock is a rectangle, specified by $\boldsymbol{W_s}$ and $\boldsymbol{H_s}$, which denotes the width and height of stock $\boldsymbol{s}$ respectively.

Let $\boldsymbol{I}$ denotes the set of all items that need to be cut and $\boldsymbol{i}$ denotes a member of $\boldsymbol{I}$. In this specific instance of the 2D SCP problem, items are rectangles, specified by $\boldsymbol{W_i}$, and $\boldsymbol{H_i}$, which denotes which the width, height of item $\boldsymbol{i}$ respectively. Each items is also associated with a demand, which is denoted by $\boldsymbol{D_i}$ for an item $\boldsymbol{i}$.

A valid cutting pattern on stock $\boldsymbol{s}$ is defined as the arrangement of some items into stock $\boldsymbol{s}$ so that it satisfies the guillotine cut condition.

To formalize the definition for a pattern as well as the conditions for a guillotine cut, for a stock $\boldsymbol{s}$, let $\boldsymbol{R} = \boldsymbol{(O, x, y)}$ be a 2 dimensional the orthogonal space where O lies in the bottom left corner of $\boldsymbol{s}$ and the two axis (x and y) runs along the width and height of the stock respectively.

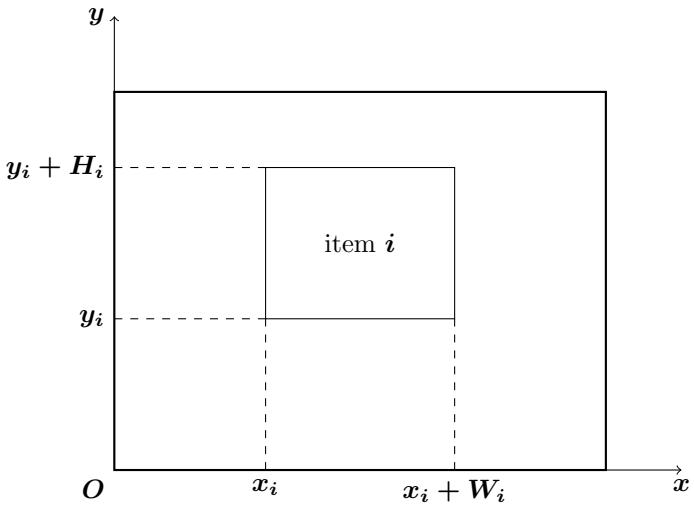

Figure 2: Visualizing orthogonal space $\boldsymbol{R}$

A cutting pattern $\boldsymbol{j}$ for the stock $\boldsymbol{s}$ can then be defined by a set of items that are included in the pattern, denoted by $\boldsymbol{I_j}$ and a sequence of points $\boldsymbol{S_j} = \{[x_i, y_i] \mid i \in I_j\}$, where each point $[\boldsymbol{x_i}, \boldsymbol{y_i}]$ representing the coordinate of the bottom left corner of item $\boldsymbol{i}$ in space R. The sequence $\boldsymbol{S_j}$ is indexed starting from 1 where $[\boldsymbol{x_{i1}}, \boldsymbol{y_{i1}}]$ denotes the coordinates of the bottom left corner of the first item in the sequence, $[\boldsymbol{x_{i2}}, \boldsymbol{y_{i2}}]$ denotes the second and so on up to $\boldsymbol{N} = |\boldsymbol{I_j}|$.

Let $\boldsymbol{Xsorted}$ be the sequence of points $\boldsymbol{S_j}$ reordered such that $\boldsymbol{x_{i1}} \leqslant \boldsymbol{x_{i2}} \leqslant ... \leqslant \boldsymbol{x_{iN}}$. Similarly, let $\boldsymbol{Ysorted}$ be the sequence of points $\boldsymbol{S_j}$ reordered such that $\boldsymbol{y_{i1}} \leqslant \boldsymbol{y_{i2}} \leqslant ... \leqslant \boldsymbol{x_{iN}}$. Both of these sequences are also indexed from 1 to $\boldsymbol{N}$.

For any 4 integers $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}$ that satisfies $\boldsymbol{1 \leqslant a \leqslant b \leqslant N}$ and $\boldsymbol{1 \leqslant c \leqslant d \leqslant N}$. $\boldsymbol{a}$ and $\boldsymbol{b}$ will act as indexes of $\boldsymbol{Xsorted}$, while $\boldsymbol{c}$ and $\boldsymbol{d}$ will act as indexes of $\boldsymbol{Ysorted}$. The set $\boldsymbol{E(a, b, c, d)} = \boldsymbol{\{Xsorted[a], ..., Xsorted[b]\} \cap \{Ysorted[c]...Ysorted[d]\}}$ then is the set of all item coordinates that fall within the bounds of $[\boldsymbol{x_{ia}}, \boldsymbol{x_{ib}}]$ for $\boldsymbol{x}$ and $[\boldsymbol{y_{ic}}, \boldsymbol{y_{id}}]$ for $\boldsymbol{y}$.

For the above figure, the formulation of the example set $\boldsymbol{E(1,\ 4,\ 1,\ 3)}$ is:

$$\boldsymbol{Xsorted} = \{[x_{i1}, y_{i1}],\ [x_{i2}, y_{i2}],\ [x_{i3}, y_{i3}],\ [x_{i4}, y_{i4}]\}$$

$$\boldsymbol{Ysorted} = \{[x_{i3}, y_{i3}],\ [x_{i1}, y_{i1}],\ [x_{i2}, y_{i2}],\ [x_{i4}, y_{i4}]\}$$

$$\boldsymbol{E(1,\ 4,\ 1,\ 3)} = \{\boldsymbol{Xsorted}[1], \boldsymbol{Xsorted}[2], \boldsymbol{Xsorted}[3], \boldsymbol{Xsorted}[4]\} \cap \{\boldsymbol{Ysorted}[1], \boldsymbol{Ysorted}[2], \boldsymbol{Ysorted}[3]\}$$

$$\Rightarrow \boldsymbol{E(1,\ 4,\ 1,\ 3)} = \{[x_{i1}, y_{i1}],\ [x_{i2}, y_{i2}],\ [x_{i3}, y_{i3}],\ [x_{i4}, y_{i4}]\} \cap \{[x_{i3}, y_{i3}],\ [x_{i1}, y_{i1}],\ [x_{i2}, y_{i2}]\}$$

$$\Rightarrow \boldsymbol{E(1,\ 4,\ 1,\ 3)} = \{[x_{i1}, y_{i1}],\ [x_{i2}, y_{i2}],\ [x_{i3}, y_{i3}]\}$$

Here $\boldsymbol{E(1,\ 4,\ 1,\ 3)}$ contains all items coordinates that falls into the grey box, i.e all item coordinates that satisfies $\boldsymbol{x} \in [\boldsymbol{x_{i1}}, \boldsymbol{x_{i4}}]$ and $\boldsymbol{y} \in [\boldsymbol{y_{i3}}, \boldsymbol{y_{i2}}]$.

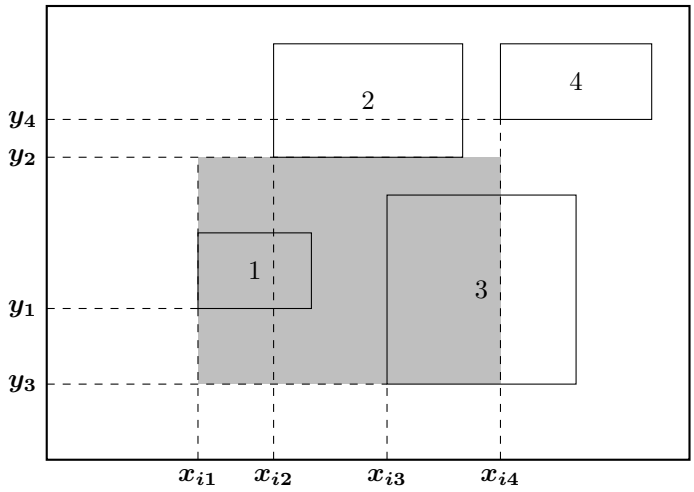

Figure 3: Example formulation of $\boldsymbol{E(1,\ 4,\ 1,\ 3)}$

Ben Messaoud, Chengbin and Espinouse (2007) [2] has proven that a given pattern is a guillotine pattern if and only if, for any 4 integers $\boldsymbol{a,\ b,\ c,\ d}$ that satisfies $\boldsymbol{1 \leqslant a \leqslant b \leqslant N}$ and $\boldsymbol{1 \leqslant c \leqslant d \leqslant N}$, at least one of these conditions is fulfilled:

1. $\boldsymbol{|E(a,\ b,\ c,\ d)| \leqslant 1}$
2. $\bigcap_{\boldsymbol{k \in E(a,\ b,\ c,\ d)}} \boldsymbol{(x_{ik}, x_{ik} + W_i)}$ is made up of at least two disjoint interval, i.e

$$\bigcap_{\boldsymbol{k \in E(a,\ b,\ c,\ d)}} \boldsymbol{(x_{ik}, x_{ik} + W_i) \neq (\min_{k \in E(a,b,c,d)} x_{ik}, \max_{k \in E(a,b,c,d)} x_{ik} + W_i)}$$

3. $\bigcap_{\boldsymbol{k \in E(a,\ b,\ c,\ d)}} \boldsymbol{(y_{ik}, y_{ik} + H_i)}$ is made up of at least two disjoint interval, i.e

$$\bigcap_{\boldsymbol{k \in E(a,\ b,\ c,\ d)}} \boldsymbol{(y_{ik}, y_{ik} + H_i) \neq (\min_{k \in E(a,b,c,d)} y_{ik}, \max_{k \in E(a,b,c,d)} y_{ik} + H_i)}$$

Let $\boldsymbol{J}$ denotes the set of all valid cutting patterns for all types of stocks, $\boldsymbol{j}$ denotes a member of $\boldsymbol{J}$ and $\boldsymbol{s_j}$ denotes the stock cutting pattern $\boldsymbol{j}$ is done on. The decision variable $\boldsymbol{x_j}$ then denotes how many stocks with cutting pattern $\boldsymbol{j}$ is used.

The 2D Cutting-Stock Problem can be stated as an ILP as follows:

$$\boldsymbol{Minimize : \sum_{j \in J} c_j \cdot x_j} \qquad \textbf{(1)}$$

$$\boldsymbol{s.t : \sum_{j \in J} a_{ij} \cdot x_j \geqslant D_i \quad \forall i \in I} \qquad \textbf{(2)}$$

$$\boldsymbol{x_j \geqslant 0 \quad and\ Integer}$$

Where (c_j) denotes the cost of the cutting pattern (j).

There are a few ways to select (c_j) depending on our goals and the applied industry, some examples are as follows:

- Choosing $c_j = 1$ turns the problem into minimizing the number of stock sheets used (1)
- Choosing $c_j = W_{s_j} * H_{s_j} - \sum_{i \in I} a_{ij} \cdot W_i \cdot H_i$ turns the problem into minimizing the total area of wasted/unused material left after cutting (2)
- For some industries, like paper or glass, where stock sheets are cut from bigger sheets or large rolls of material for standardization c_j can also represents the cost of cutting that stock size (3).

Here condition (1) enforces the minimization of waste with x_j being the decision variables, while condition (2) enforces the demands for each item be met (we allow a cutting pattern to exceed this demand). Typically, condition (1) is sufficient, as condition (2) requires more computation, but the end result is not guaranteed to be better.

The non-rotatable item condition imposed for this formulation makes it suitable for industries that care about the orientation of the items on the stock (like paper or wood) due to the stock having specific patterns on the stock that we want to preserve. For industries that allow 90-degree rotations for certain items, i.e the orientation of the items do not matter (like glass or steel), the formulation can be modified slightly to satisfy this condition.

Suppose the item that can rotate is i^*. The set of all items I can include another item that is the rotated version of i^*, i.e with width H_{i^*} and height W_{i^*}. Then for every cutting pattern $j \in J$, the variable a_{ij} should reflects the total count of both orientations in the cutting pattern j.

It is also important to view the dual problem to the LP relaxation of this formulation, which is stated as follows:

$$Maximize: \sum_{i \in I} D_i \cdot u_i$$

$$s.t: \sum_{i \in I} a_{ij} \cdot u_i \leqslant c_j \quad \forall j \in J$$

$$u_i \geqslant 0$$

u_i represents the "shadow price" of each item i. u_i plays an important role in determining if optimality is reached using column generation in the following section.

4.2 Column generation and the 2D knapsack problem

The problem with the standard formalization above is that it requires knowing set J in advance, whose length grows very fast as our stock size increases.

Here we can utilize column generation. Starting with an initial $\hat{j} \subseteq J$, and adding a new cutting pattern j^* each iteration, expanding $\hat{j}$ until optimality is reached. The procedures for column generation is as follows:

- Solves the LP relaxation of the problem (with $\hat{j}$ to be the set of valid cutting patterns used) using the Simplex method to obtain both the optimal solution X^*; along with the solution to the dual problem (i.e the "shadow price" of each item), denoted by $u_i^* \ \forall i \in I$. The "shadow price" represents the value of adding 1 more item of that type to the overall count.
- Generate a new valid cutting pattern j^* with maximal value.
- If j^* would lower the value of the cost function of the original ILP, add it to $\hat{j}$
- Let j^* be the entering variable in the new basis, continue the algorithm with the new $\hat{j}$.
- Once an optimal $\hat{j}$ is reached, i.e the adding j^* would actually increases the value of the cost function, we start branching from X^* to solve for the integer case.

The generation of a new cutting pattern is a 2D case of the knapsack problem. The general case of the problem can be stated as follows:

Given a set of items I, each with a value denoted u_i^* for $i \in I$. Maximize the value of items in a given container of specified size.

The formalization of the problem is:

$$Maximize : \sum_{i \in I} u_i^* \cdot a_i$$

s.t: the items still fit in the container, determine by some linear relation, in this case the number of items still fit on the 2D stock

$$a_i \geqslant 0 \quad and\ Integer$$

With a_i denotes the number of item i inside the container

Solving this problem can also be troublesome as the general 2D problem requires generating and validating a valid cutting patterns in a reasonable amount of time. There have been methods devised to exhaustively search the possible state space and arrive on an optimal or near optimal solution[8]. But our group focuses on the formulation of the problem where some constraints have been placed.

4.2.1 2 stages guillotine (with trimming)

Limiting the number of stages for guillotine cuts significantly reduces the number of valid cutting patterns. Guillotine cuts done in 2 stages with trimming is typically in the industrial cutting standard for glass, metal, etc. and typically follows the steps of:

- First-stage guillotine cuts split the stock into multiple strips along the longer axis.
- Second stage guillotine cuts split the strips into smaller pieces using cuts along the remaining axis.
- Third stage guillotine cuts trims the pieces that needs to be even smaller.

For 2 stages guillotine cuts, P. C. Gilmore and R. E. Gomory (1965)[6] showed that the problem of generating a stock j^* is reducible to two separate 1 dimensional knapsack problem, corresponding to the 2 stages of the cutting process.
(Without loss of generality the following formulation has the assumption of $W_s \geqslant H_s$)

Stage 1:
For the first stage of cutting, the stock s is cut into strips along the width of the stock. For this stage, for each unique item height, a strip of that height is made. A strip is constructed by stacking items of equal or lesser height than the strip's height end to end width wise.

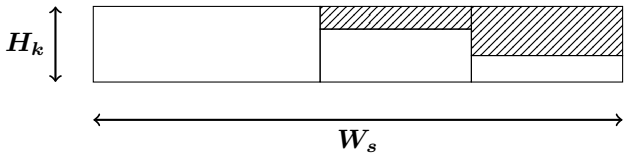

Figure 4: Visualization of a strip

Let k denotes the strip that is being constructed, whose height is H_k. The set I_k then denotes the set of all items whose height is less than or equal to strip k's height.

The calculation for a_{ik}, which is the amount of item i being stacked in strip k, is stated as an ILP as follows:

$$Z_k = maximize : \sum_{i \in I_k} u_i^* \cdot a_{ik}$$

$$s.t : \sum_{i \in I_k} W_i \cdot a_{ik} \leqslant W_s$$

$$a_{ik} \geqslant 0 \quad and\ Integer$$

Z_k is then the optimum value for strip k.

Stage 2:
For stage 2 requires the calculation of one more 1 dimensional knapsack problem to choose how many of each strips to include in the cutting pattern j^*, stacked end to end height wise.

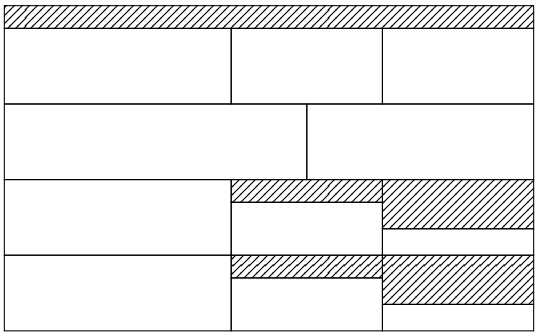

Figure 5: Visualization of the stacking of strips

Let $\boldsymbol{K}$ be the set of all strips generated from stage 1 and $\boldsymbol{k}$ be a member of set $\boldsymbol{K}$. The calculation of $\boldsymbol{a_{kj^*}}$ follows roughly the same format as the formulation in stage 1 and can be stated as follows:

$$\boldsymbol{Z_{j^*} = maximize \sum_{k \in K} Z_k \cdot a_{kj^*}}$$

$$\boldsymbol{s.t : \sum_{k \in K} H_k \cdot a_{kj^*} \leqslant H_s}$$

$$\boldsymbol{a_{ki} \geqslant 0 \quad and\ Integer}$$

One important note here is since the value is the same, the order of the strips do not matter. Once the amount of each strip in $\boldsymbol{j^*}$ is know, the actual cutting pattern is produced by stacking the strip in whatever order.

We also need to decide if the generated cutting pattern $\boldsymbol{j^*}$ will indeed reduce the value of our original cost function. With the inclusion of $\boldsymbol{j^*}$, the cost function becomes:

$$\boldsymbol{Minimize : (c_{j^*} \cdot x_{j^*}) + \sum_{j \in old\hat{j}} c_j \cdot x_j} \qquad \textbf{(1)}$$

To determine if the new cost function is lower than the original, we want the partial derivative of this new cost function with respect to the added variable $\boldsymbol{x_{j^*}}$. In this specific case, this simplifies to:

$$\boldsymbol{c_{j^*} - Z_{j^*}}$$

If ever the reduced cost is positive or zero, that means the current set of cutting patterns for stock j is optimal, adding more cutting patterns would make the cost function increases or causing no changes to it. i.e, No new cutting pattern can decrease the objective function of the original Cutting-stock problem.

Since we have multiple sizes or types of stocks, we repeat this procedure for every types and stop only once the generated new cutting pattern j^* for every type would not reduce the cost function for the original ILP.

One further note is that the above formulation directly follows the steps mentioned for the real life cutting procedures, which prioritizes the longer axis as the first axis to split into strips. This may misses out on valid cutting patterns that prioritizes splitting the shorter axis first. This can be easily fixed by doing more computations, specifically all the above computations for the generation of a valid cutting pattern of a stock type s again with the first stage splitting the shorter axis first. This may be unnecessary for most real life variations of the 2D SCP however. As absolute optimality is not required, the solution that prioritizes splitting the longer axis first is good enough for most situations.

4.2.2 general 3 stages guillotine

One important distinction between the general 3 stages guillotine case from the 2 stages with trimming is that the third stage can not just trim the heights of each "column" of items, it can also split that "column" into more of the desired types.

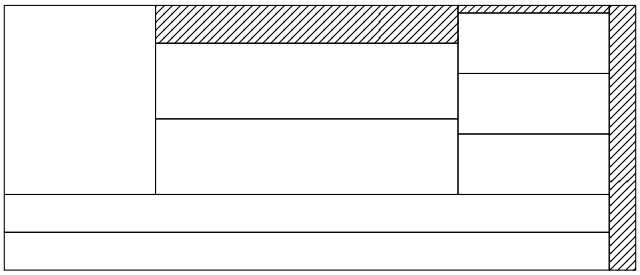

Figure 6: A situation where a stage 3 cut can split a "column"

To accommodate for this, only the formulation for stage 1 needs to be changed. Instead of stacking only 1 item end to end, the formulation can make a stack height wise of the same item up to strip height, then stack these same-item-stack width wise as before. Since the problem is a maximization problem and all coefficients are not negative ($u_i^* \geqslant 0 \ \forall i \in I$), i.e adding more items will always increase the objective function. Thus, we do not need to solve another ILP for each same-item-stack and all the same-item-stack can be generated by stacking the maximum number of that item to the strip height. The number of items i per same-item-stack in a strip k with height H_k is given by:

$$M_{ik} = floor(H_k / H_i)$$

The cost function of the formulation for stage 1 is thus changed to:

$$Z_k = maximize : \sum_{i \in I_k} u_i^* \cdot M_{ik} \cdot a_{ik}$$

4.2.3 Extending the formulation to higher dimensions

Interestingly, the formulation for column generation in the above sections can be apply recursively to extend to more dimensions.

With stock of $\boldsymbol{K}$ dimensions, the cutting process can be done in $\boldsymbol{K}$ recursive calls, each solving the problem on a lower axis. The procedure is as follows:

- For an axis $\boldsymbol{a}$, For every items, the set $\boldsymbol{I_a} \in \boldsymbol{I}$ representing a slice along the axis $\boldsymbol{a}$ containing the item $\boldsymbol{i}$ and all items whose dimension along axis $\boldsymbol{a}$ is less than or equal to item $\boldsymbol{i}$'s dimension along axis $\boldsymbol{a}$.
- For every set $\boldsymbol{I_a}$, solve the problem in 1 dimension lower, with $\boldsymbol{I_a}$ being the set of items being used and ignoring axis $\boldsymbol{a}$.
- Solve 1 more 1 dimensional knapsack problem, optimizing the value of the current slice along axis $\boldsymbol{a}$ by stacking smaller slices (dimension wise) generated in the step above along this axis.

5 Case study

5.1 Problem Description

This study stems from an industrial project undertaken at Silkar Mining Corporation, a company with a long-standing presence in the natural stone industry. Established in 1994 in Bilecik, Turkey, Silkar operates under the AKDO brand and has grown to become a prominent player in its field. With a focus on providing high-quality products, the corporation specializes in natural stones such as marble, granite, and ceramics. Over the years, it has built a strong reputation for innovation and reliability, earning its place as a leader in the natural stone sector.

Silkar Corporation serves a diverse clientele that spans both domestic and international markets. Within Turkey, the company provides products to various industries, including the service and manufacturing sectors. Beyond its home market, Silkar exports its products to over 35 countries worldwide. Its international reach extends to regions such as the United States, Europe, North Africa, and the Far East, reflecting its commitment to meeting the demands of a global customer base. This wide distribution network has solidified Silkar's position as a trusted supplier in the natural stone industry.

The company's core operations involve processing raw materials like marble, granite, and ceramics, which are sourced from their natural origins. These materials arrive at the factory in the form of large blocks, ready for processing into finished products. The transformation of these raw materials into high-quality goods exemplifies Silkar's expertise in blending traditional craftsmanship with modern technology. As shown in Figure 7, the raw blocks undergo various stages of refinement to meet the exacting standards of customers around the world.

For those seeking more information about Silkar Mining Corporation, additional details can be found on the company's official website at this link. This resource provides further insights into their operations, product offerings, and contributions to the natural stone industry. Through its dedication to quality and customer satisfaction, Silkar continues to uphold its reputation as a pioneer in the field of natural stones.

Figure 7: The raw materials – marble blocks

The blocks are first processed to achieve uniform three-dimensional shapes with precise dimensions. These dimensionally accurate blocks are then sliced layer by layer to produce multiple two-dimensional objects. The layers are typically cut at specific, consistent heights. The resulting two-dimensional objects, with their designated lengths and widths, are referred to as panels.

The number of cuts made on a block of type $\boldsymbol{h}$ determines the number of panels produced from it, denoted as $\text{num}_{\boldsymbol{h}}$ in our notation. Each panel of type $\boldsymbol{h}$ has two dimensions: length ($\boldsymbol{L_h}$) and width ($\boldsymbol{W_h}$). Figure 8 illustrates the process, showcasing how panels with two-dimensional measurements are derived from the initial three-dimensional blocks.

Figure 8: The two dimensional panels

From these panels, smaller objects—referred to as items—are produced, subject to the limited availability of the panels. Each of these small items is assigned to customers who require several of them to assemble their final products. Figure 9 depicts a small cut item with specific dimensions (length and width) and illustrates the arrangement of such items to create a final product, such as floor coverings, wall tiles...

For any given period, the company's objective is to maximize the total profit. This profit is calculated as the difference between the total revenue generated from the sale of small items and the total cost incurred from the panels used to produce these items. Achieving this goal requires carefully balancing production to meet the specified customer demand for each type of item while ensuring that the availability of panels is not exceeded. The optimization process must account for the constraints of limited resources and customer satisfaction to ensure efficient and profitable operations.

In the following section, we will use data from a random customer's order and apply the tools mentioned earlier in this report to determine the nearly optimal solution.

Figure 9: The final products

5.2 Optimization Problem

No.	Width	Length	Demands
1	60	79	14
2	60	113	50
3	60	109	28
4	38	60	28
5	35	30	10
6	35	60	7
7	30	79	30
8	30	30	5
9	30	38	4
10	30	109	5

Table 4: Case study - Customer's Order

As can be seen in Table 4, the customer has specified a variety of item types, each with different dimensions and quantities required. This illustrates the complexity and diversity of the customer's needs. In response to these requirements, we have compiled a table of stock panels, which are the available materials from which these items will be cut. The following table provides a detailed overview of the stock panels, which will be used to fulfill the customer's order based on the specified dimensions and quantities.

No.	Width	Length	Quantity
1	130	205	10
2	135	165	17
3	135	210	15
4	135	250	4
5	140	270	5

Table 5: Case study - Available panels' Data

5.3 Solution

As demonstrated in the modeling section, while it is theoretically possible to solve the problem manually, the process is exceedingly intricate and impractical for larger or more complex instances. This is why we have chosen to leverage various computational tools to arrive at a solution more efficiently. To address the 2D cutting stock problem, we have developed specialized algorithms written in Python, which serve as our primary tools for tackling this challenge. Rather than attempting to manually solve the entire problem step by step, we will instead rely on these algorithms to generate potential solutions. Additionally, we plan to compare the results produced by our algorithms with those generated by other tools and methods that are readily accessible online, ensuring a robust evaluation of the effectiveness and accuracy of our approach.

5.3.1 Our algorithms

Here are the results after running our two main algorithms (the combination heuristic and the RL agent) through this environment:

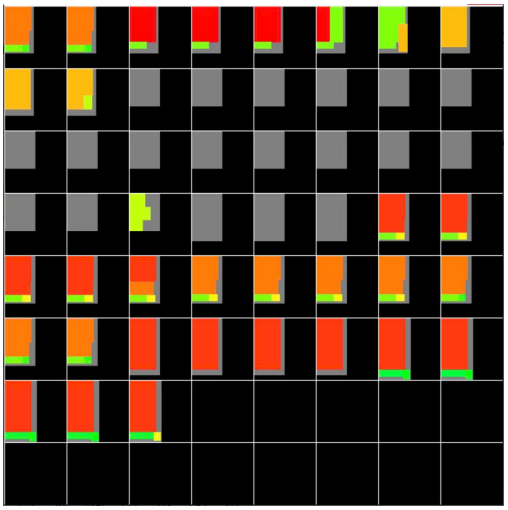

Figure 10: Combination heuristic cutting patterns for this case study

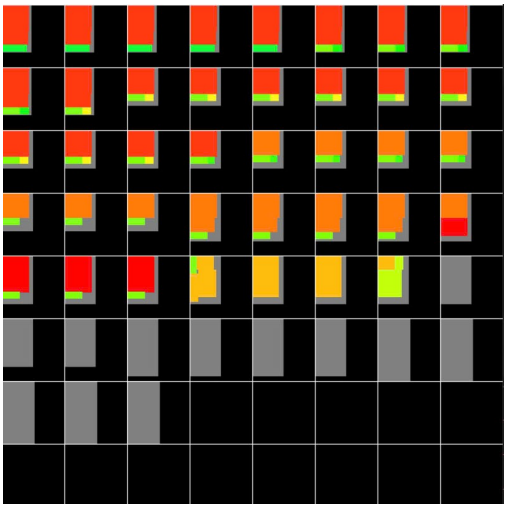

Figure 11: RL agent's produced cutting patterns for this case study

Here is the color map of the items used for the illustration:

Item no.	Color
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Table 6: Item to color mapping used to display cutting patterns

Overall, here are the results of the two algorithms:

Combination heuristic result:

Stock sizes	Stock count	Unused
205 x 130	10	0
165 x 135	01	16
210 x 135	12	3
250 x 135	04	0
270 x 140	05	0

- Total stocks used: 32
- Overall waste rate: 0.1994
- Overall fitness: 0.8006

RL agent result:

Stock sizes	Stock count	Unused
205 x 130	10	0
165 x 135	17	0
210 x 135	12	3
250 x 135	00	0
270 x 140	00	0

- Total stocks used: 39
- Overall waste rate: 0.2243
- Overall fitness: 0.7757

5.3.2 Online tool

For comparisons, we will additionally use a tool named "optiCutter" to solve the case study problem. The website for this tool can be found at this link.

Based on the data provided in the previous section of the report, we have the input data shown in Figure 10 and Figure 11.

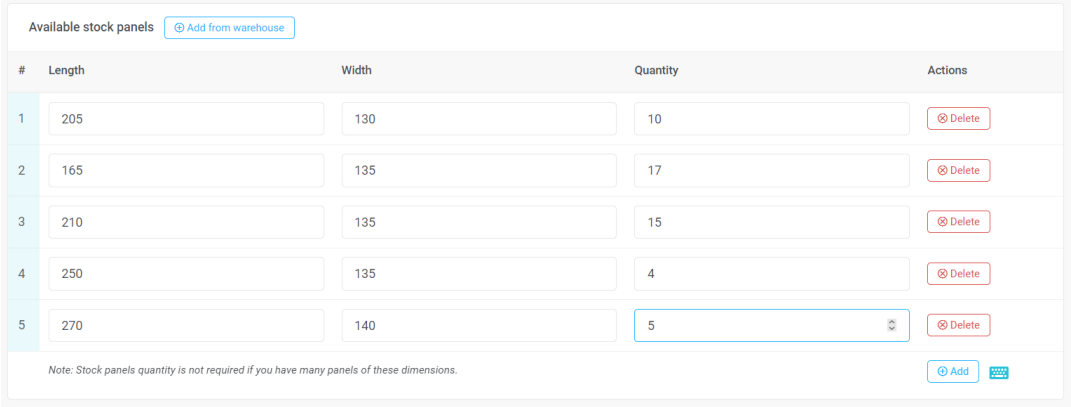

Figure 12: Input to optiCutter - List of available Stock panels

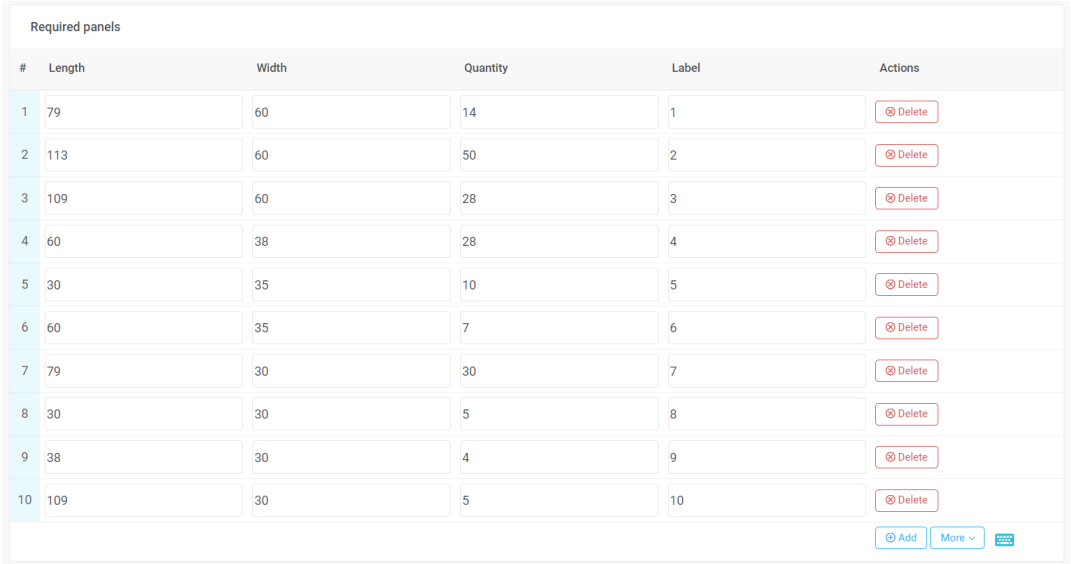

Figure 13: Input to optiCutter - List of Ordered items

The outcomes of the analysis will be systematically summarized and illustrated in Figure 14 for ease of reference. Furthermore, a comprehensive collection of visual representations depicting the arrangement, organization, and conceptualization of the cutting patterns will be carefully compiled and made accessible via this link.

Required stocks

Stock dimensions	Quantity
205 x 130	10
165 x 135	8
210 x 135	8
250 x 135	4
270 x 140	3
Total	33

Description / Material	Stone Panels
Total panels area (Quantity)	774,030 (181)
Used stocks total area (Yield)	919,900 (84.143 %)
Total cutting layouts	22
Total cuts length (Quantity)	24,935 (262)

Figure 14: Opticutter's statistics on the its solution to this case study

From Figure 14, we can provide a more detailed evaluation of this online tool's results as follows:

- Overall waste rate: 0.1586
- Overall fitness: 0.8414

Compare to our algorithms, our group found that while the best results from our combination heuristic do give a lower count for the amount of used stock compared to this online tool (32 stocks to 33). We sacrifices in terms of overall fitness and waste rate, where the solution provided from the online tool performs better on both of these metrics. This comes down to the differences in what metrics is being optimized and depending on the real life circumstances, a lower stock count may be preferable to a lower overall wasted material.

6 Evaluate the algorithms

6.1 Frameworks

We will use the provided gymnasium library to provide randomized sets of items and stock sizes as a benchmark to compare the algorithms proposed in this report based on the following evaluation criteria:

- Best waste rate: The minimum rate of wasted area to used area across all results in a batch, determined by the formula below:

$$\boldsymbol{Waste\ rate} = \frac{\boldsymbol{Total\ unused\ area\ of\ all\ stocks}}{\boldsymbol{Total\ area\ of\ all\ cut\ stocks}}$$

  This value represents how much area on the stock gets wasted. The closer this value to being 0, the better. When it is 0, no material gets wasted/unused.

- Average waste rate: The average waste rate of wasted area to used area across all results in a batch.

- Best fitness: The best fitness value across all results in a batch, determined by the formula below:

$$\boldsymbol{Fitness} = \frac{\boldsymbol{Total\ area\ of\ all\ cut\ items}}{\boldsymbol{Total\ area\ of\ all\ cut\ stocks}}$$

  This value represents how closely the items "fit" on the stocks in a given pattern. The closer this value to being 1, the better. When it is 1, the items fully covers the stocks.

- Average fitness: The average fitness value across all results in a batch.

- Best time: The best execution time across all results in a batch, measured in seconds (s).

- Average time: The average execution time across all results in a batch, measured in seconds (s).

6.2 Results after execution

The following tables presented the results after bench marking 3 different algorithms using 5 batches of randomized environments. Each batch contains 10 different seeds for environment generation, totaling 50 different environment set ups.

No.	Best waste rate	Average waste rate	Best fitness	Average fitness	Best time	Average time
1	0.1714	0.3373	0.8537	0.7602	0.9387	19.1134
2	0.1477	0.2924	0.8713	0.7814	1.0790	46.1255
3	0.1175	0.8661	0.8949	0.6820	0.0040	77.9725
4	0.1188	0.3385	0.8938	0.7587	0.1318	18.9791
5	0.1808	0.2673	0.8469	0.7912	0.2565	17.9854

Table 7: Greedy Algorithm (First fit) bench mark results

No.	Best waste rate	Average waste rate	Best fitness	Average fitness	Best time	Average time
1	0.0561	0.1466	0.9469	0.8782	1.6775	9.4369
2	0.0478	0.0948	0.9544	0.9154	1.3211	13.8800
3	0.0348	0.2466	0.9664	0.8581	0.1556	12.4730
4	0.0409	0.1138	0.9607	0.9002	0.3321	7.4599
5	0.0538	0.1092	0.9490	0.9030	0.5570	8.2988

Table 8: Combination Heuristic Algorithm bench mark results

No.	Best waste rate	Average waste rate	Best fitness	Average fitness	Best time	Average time
1	0.0804	0.2132	0.9256	0.8365	2.9585	37.5355
2	0.0565	0.1955	0.9465	0.8517	1.4830	47.7499
3	0.0396	0.7459	0.9619	0.7620	0.1854	54.3464
4	0.0829	0.1590	0.9235	0.8681	0.6262	39.5981
5	0.0730	0.1647	0.9320	0.8629	0.6247	39.4916

Table 9: Reinforcement Learning agent bench mark results

To compare these performances, our group plots the average waste rate, average fitness and average time of the 3 algorithms together:

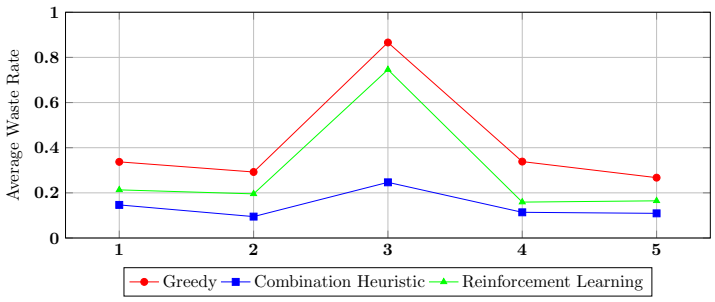

Figure 15: Comparison of Average Waste Rates Across Algorithms

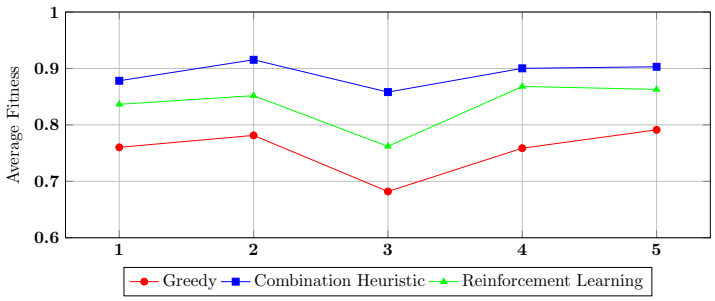

Figure 16: Comparison of Average Fitness Across Algorithms

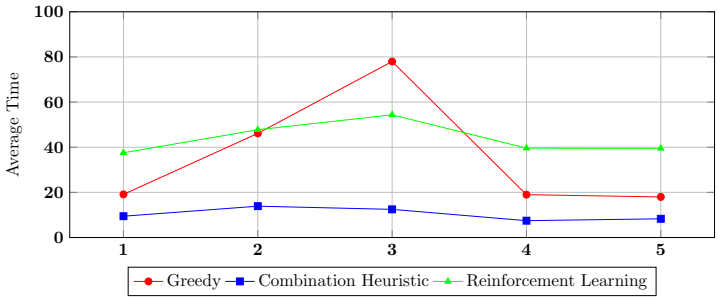

Figure 17: Comparison of Average Time Across Algorithms

6.3 Evaluation

In this bench mark test, combination heuristic performs consistently better than the other two, with RL following close behind on every batches. This may due to our RL model unable to choose the stock and items itself and delegates this task to a similar heuristic. More training time for the RL model also should provides better results. Comparatively speaking, this result is not the optimal results and more improvements can be done in the future.

7 Conclusion

7.1 Source code, data and related materials

- All materials and everything related to the source code can be found at this github branch of the original project filesthis path.
- The raw data that we use to serves as the foundation for our analysis be found at this path.

7.2 Afterword

2D CSP and together with it the 2D knapsack problem are two NP-hard problems that have real-life applications in many industries. Our approaches here in this big exercise are largely approximation based as finding an optimal solution for big problem is not feasible. Moving forward, the formulation of the problem into ILP along with column generation has been adapted several times to solve real-world problems [3] [4]. While some papers have attempted other techniques for solving the problem like exhaustive branch and prune application on the possible state space that provides a near optimal solution of the problem[8] or adapting a heuristic approach [14]. Our group's RL approach also seems promising if given more training time and a more extensive reward function. The RL approach is also extendable to other environments whose restrictions is different than the standard requirements [11]. It is interesting to explore further the capabilities and applications of RL and machine learning techniques as a whole to the problem. Overall, the problem and its variations opens to many approaches with varying degrees of accuracy.

References

[1] Umutcan Ayasandır and Meral Azizoğlu. "Two-dimensional cutting stock problem with multiple stock sizes". In: *International Journal of Manufacturing Technology and Management* 38.2 (2024), pp. 95–125. DOI: 10.1504/IJMTM.2024.137486. eprint: https://www.inderscienceonline.com/doi/pdf/10.1504/IJMTM.2024.137486. URL: https://www.inderscienceonline.com/doi/abs/10.1504/IJMTM.2024.137486.

[2] Said Ben Messaoud, Chengbin Chu, and Marie-Laure Espinouse. "Characterization and modelling of guillotine constraints". In: *European Journal of Operational Research* 191.1 (2008), pp. 112–126. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2007.08.029. URL: https://www.sciencedirect.com/science/article/pii/S0377221707009083.

[3] G.F. Cintra et al. "Algorithms for two-dimensional cutting stock and strip packing problems using dynamic programming and column generation". In: *European Journal of Operational Research* 191 (Nov. 2008), pp. 61–85. DOI: 10.1016/j.ejor.2007.08.007.

[4] Fabio Furini et al. "A column generation heuristic for the two-dimensional two-staged guillotine cutting stock problem with multiple stock size". In: *European Journal of Operational Research* 218 (Apr. 2012), pp. 251–260. DOI: 10.1016/j.ejor.2011.10.018.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. First Edition. W. H. Freeman, 1979. ISBN: 0716710455. URL: http://www.amazon.com/Computers-Intractability-NP-Completeness-Mathematical-Sciences/dp/0716710455.

[6] P. C. Gilmore and R. E. Gomory. "Multistage Cutting Stock Problems of Two and More Dimensions". In: *Operations Research* 13.1 (1965), pp. 94–120. ISSN: 0030364X, 15265463. URL: http://www.jstor.org/stable/167956 (visited on 12/04/2024).

[7] Khadija Hadj Salem et al. "Mathematical models for the two-dimensional variable-sized cutting stock problem in the home textile industry". In: *European Journal of Operational Research* 306.2 (2023), pp. 549–566. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2022.08.018. URL: https://www.sciencedirect.com/science/article/pii/S0377221722006610.

[8] Manuel Iori et al. "Exact solution techniques for two-dimensional cutting and packing". In: *European Journal of Operational Research* 289.2 (2021), pp. 399–415. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2020.06.050. URL: https://www.sciencedirect.com/science/article/pii/S0377221720306111.

[9] Andrea Lodi and Michele Monaci. "Integer linear programming models for 2-staged two-dimensional Knapsack problems". In: *Math. Program.* 94 (Jan. 2003), pp. 257–278. DOI: 10.1007/s10107-002-0319-9.

[10] Rita Macedo et al. *2D Cutting Stock Optimization Software Survey*. Jan. 2008.

[11] Mateus Martin et al. "Models for two-and three-stage two-dimensional cutting stock problems with a limited number of open stacks". In: *International Journal of Production Research* 61 (Apr. 2022). DOI: 10.1080/00207543.2022.2070882.

[12] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347.

[13] François Vanderbeck. "A Nested Decomposition Approach to a Three-Stage, Two-Dimensional Cutting-Stock Problem". In: *Management Science* 47 (June 2001), pp. 864–879. DOI: 10.1287/mnsc.47.6.864.9809.

[14] Dianjian Wu and Guangyou Yang. "A Heuristic Approach for Two-Dimensional Rectangular Cutting Stock Problem considering Balance for Material Utilization and Cutting Complexity". In: *Advances in Materials Science and Engineering* 2021.1 (2021), p. 3732720. DOI: https://doi.org/10.1155/2021/3732720. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1155/2021/3732720. URL: https://onlinelibrary.wiley.com/doi/abs/10.1155/2021/3732720.