HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY

COMPUTER ENGINEERING

MATHEMATICAL MODELING (CO2011)

Assignment

"Cutting sotck problem"

Class: CC04

Lecturer: Tran Hong Tai

Student : Pham Van Bach 2252057

Ho Chi Minh City, November 11th, 2024

Contents

1 Introduction

The cutting stock problem (CSP) has its origins in the groundbreaking work of L.V. Kantorovich in 1939, initially arising in the paper industry. The problem involves cutting large rolls or sheets of fixed dimensions into smaller pieces to fulfill specific orders while minimizing waste. In 1951, before the widespread use of computers, Kantorovich and V.A. Zalgaller proposed addressing this challenge using linear programming. Their innovative approach introduced the column generation method, which became a key technique in solving large-scale optimization problems. Over time, CSP has been applied to a wide range of industries, including steel, textiles, and aluminum, and has evolved to address more complex variants such as the two-dimensional cutting stock problem (2D-CSP).

The 2D-CSP extends the classic problem to a two-dimensional space, where large sheets of material must be cut into smaller rectangular pieces to meet demand. This adds complexity due to additional constraints such as guillotine cuts, material orientation, and layout optimization. The primary goal remains the same: to minimize waste and cutting costs while maximizing material utilization. The 2D-CSP is particularly relevant in industries where efficient use of resources is critical for reducing costs and environmental impact.

In this project, we focus on developing a solution to the 2D-CSP using advanced optimization techniques. By leveraging modern algorithms and computational tools, our aim is to tackle the inherent challenges of this NP-hard problem. The methods explored in this project not only contribute to the theoretical understanding of 2D-CSP but also have practical implications for industries seeking to improve efficiency and sustainability in their operations.

2 Acknowledge

2.1 Algorithmic methods

Linear Programming for 2D-CSP: Gilmore and Gomory's (1965) foundational work extended to two dimensions, where they addressed cutting stock problems under guillotine constraints. Their exact algorithm for generating patterns minimized waste and formed the basis for modern 2D cutting approaches.
Nested Decomposition Approach: Vanderbeck (2001) introduced a nested decomposition method for multi-stage 2D-CSP, enabling the solving of problems with complex stock and demand patterns by breaking them into manageable subproblems.
Branch-and-Cut for 2D Problems: Christofides and Hadjiconstantinou (1995) advanced 2D-CSP by applying branch-and-cut algorithms. This method efficiently handled problems involving constraints on maximum cuts and complex item arrangements.

2.2 Heuristic methods

Iterative Pattern Generation: Gradisar et al. (1999) proposed a sequential heuristic procedure tailored for 2D-CSP. Their method emphasized iteratively generating cutting patterns to optimize layout and minimize trim loss, particularly for rectangular items.
Metaheuristics with Pattern Adaptation: Umetani et al. (2003) applied metaheuristic strategies like simulated annealing and genetic algorithms to 2D-CSP. They coupled these with adaptive pattern generation to dynamically refine cutting layouts.
Practical Heuristic for Guillotine Cuts: Silva et al. (2011) developed a heuristic for 2D guillotine cutting problems, prioritizing simplicity and computational efficiency. Their approach generated near-optimal solutions for industrial use cases involving rectangular items.

2.3 Combined methods (Algorithm and Heuristics)

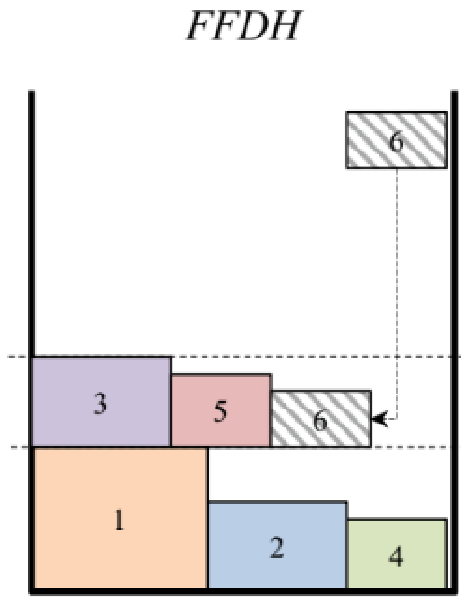
Column Generation with Metaheuristics: Vance et al. (1994, 1998) integrated column generation techniques with heuristic methods to solve large-scale 2D-CSP. By iteratively improving cutting patterns, their method addressed both efficiency and scalability challenges.
Hybrid Optimization Framework: Dyckhoff (1990) outlined a framework for combining exact and heuristic methods for 2D-CSP. By leveraging both pattern-oriented and item-oriented strategies, this approach achieved robust solutions for diverse problem types.

Branch-and-Cut-and-Price for 2D-CSP: Belov and Scheithauer (2007) extended their branch-and-cut-and-price algorithm to two dimensions, effectively handling problems with heterogeneous stock sizes and complex layout requirements.

3 Algorithm

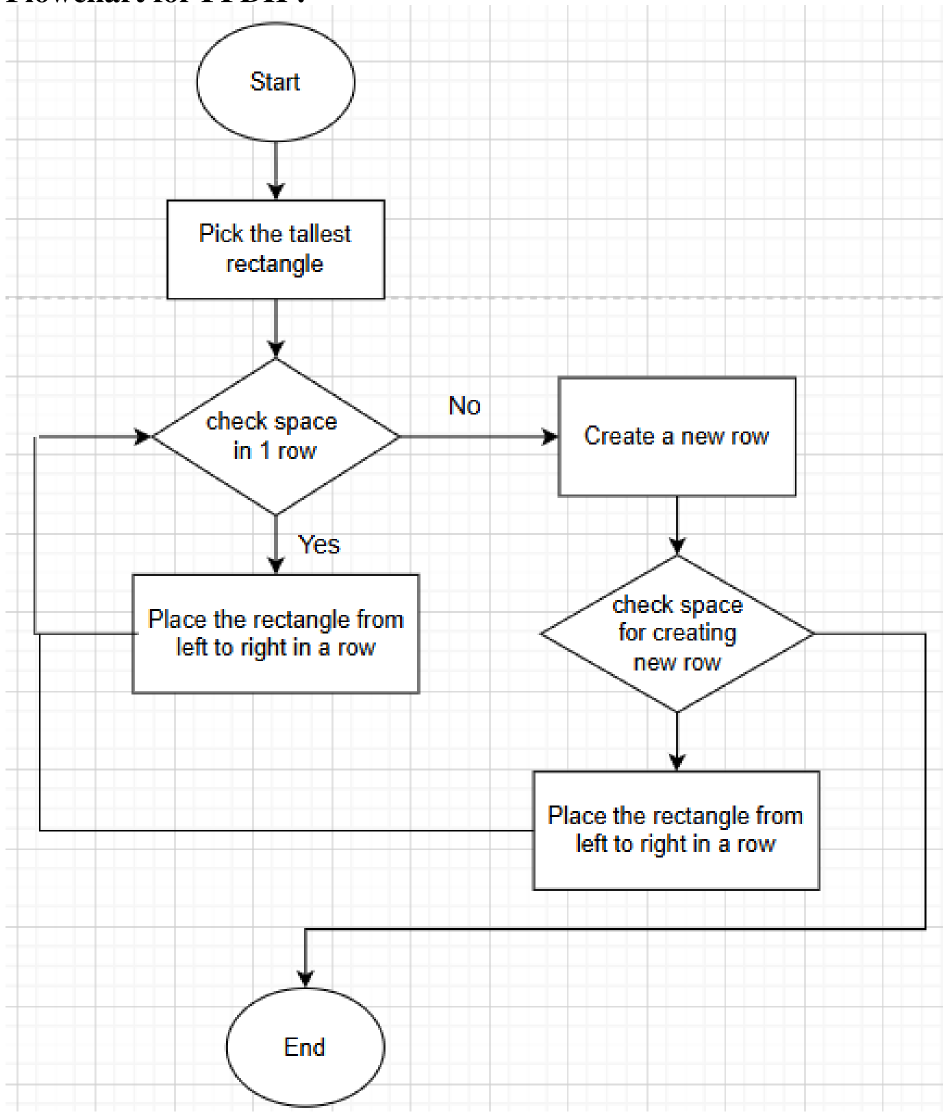
The 2D Cutting Stock Problem (CSP) is a combinatorial optimization challenge encountered in manufacturing and industrial applications. It involves cutting a large material, such as a sheet, roll, or plate (referred to as ”stock”), into smaller, predefined shapes and sizes (called ”products”) to fulfill specific demand quantities. The main objective of the CSP is to minimize the waste material left over after the cutting process. Waste occurs when parts of the stock are not utilized due to the inability to fit products into the available space. Since this problem is NP-hard, finding an exact solution is computationally expensive, especially for large-scale problems. As a result, various heuristic approaches are used to find near-optimal solutions efficiently, with two common heuristics being First Fit Decreasing Height (FFDH) and Best Fit Decreasing Height (BFDH).

3.1 First Fit Decreasing Height (FFDH)



In the First-Fit Decreasing Height (FFDH) algorithm, a list of rectangles is first ordered by their height in non-increasing order. This sorting ensures that the tallest rectangles are placed first, which can help reduce the likelihood of leftover space in the stock sheets. Rectangles of equal height maintain their relative order in the list, as they are not reordered beyond the height-based sorting. Once the rectangles are sorted, the packing process begins by placing each rectangle in the first available position, starting from the bottom of the strip. The algorithm searches level by level, from the lowest point upwards, to find a spot where the rectangle can fit. If no suitable position is found within the current levels, a new level is created above the previous ones, and the rectangle is placed there, still left-justified. This procedure contrasts with the Next-Fit Decreasing Height (NFDH) algorithm, where previously packed levels may not be revisited. In NFDH, once a rectangle does not fit in the current level, it is placed in a new level, without searching for a fit within earlier packed levels. The primary difference between FFDH and NFDH is that FFDH actively looks for available space across existing packed levels before resorting to the creation of new levels, which can lead to a more compact packing arrangement and potentially less unused space.

Flowchart for FFDH :


Start
Pick the tallest rectangle
check space in 1 row
No
Create a new row
Yes
Place the rectangle from left to right in a row
check space for creating new row
Place the rectangle from left to right in a row
End

Advantages:

- FFDH is easy to implement and understand. It does not require complex calculations or data structures.

- The algorithm is generally faster than more complex methods like Integer Linear Programming (ILP) or branch-and-bound because it only needs to perform a simple search for each rectangle.

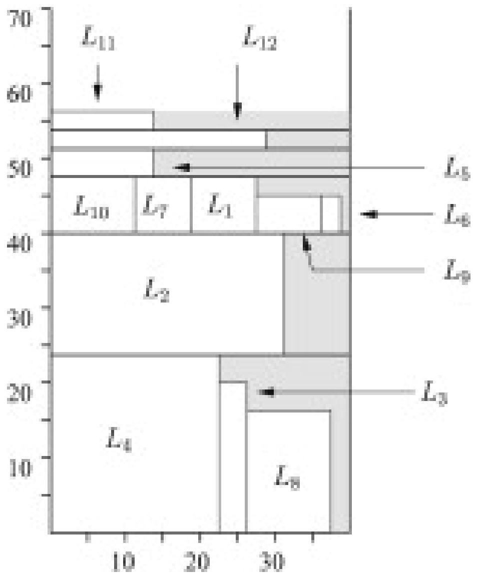
- FFDH provides a good heuristic solution, even for large problems. While not optimal, the results are typically near-optimal and can be obtained in a reasonable amount of time.

- For larger instances of the 2D-CSP, the algorithm performs significantly better compared to exact methods like ILP, as it avoids exhaustive search.
- By sorting rectangles in non-increasing order, the algorithm tends to pack larger items first, which can prevent large gaps from forming early in the packing process.

Disadvantages:

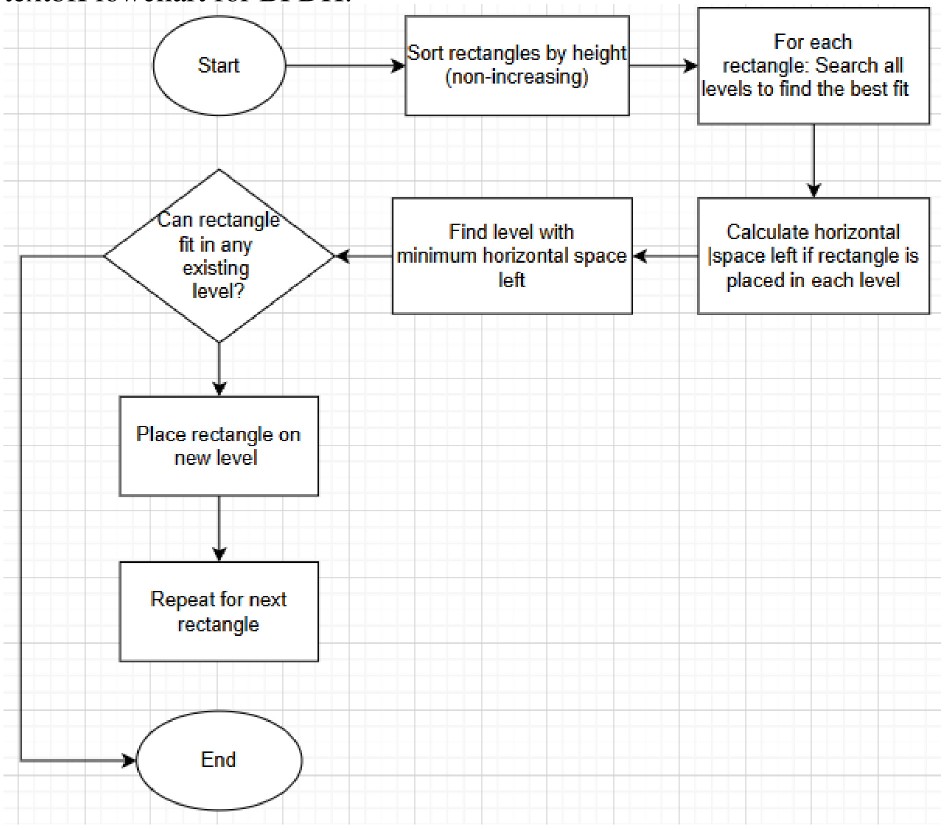
- FFDH may not always produce the most efficient packing solution. It tends to create many small gaps and unused spaces, particularly in scenarios where the stock sheet dimensions don't align well with the items.
- The performance of FFDH can be sensitive to the initial sorting of items.
- Once an item is placed on a level, it cannot be moved to a different level. This can lead to inefficient use of available space.
- While FFDH reduces the creation of new levels, it might still accumulate waste in some cases because it doesn't consider the residual space available in the previous levels before placing an item.

3.2 Best-Fit Decreasing Height



The Best-Fit Decreasing Height (BFDH) algorithm shares similarities with the First-Fit Decreasing Height (FFDH) algorithm, with the key difference being in the placement strategy of the rectangles. In BFDH, the rectangles are also sorted in non-increasing order of their height, ensuring that taller rectangles are packed first. However, instead of placing each rectangle in the first available level, BFDH searches through all the existing levels to find the one that leaves the least amount of unused horizontal space when the rectangle is placed on it. The rectangle is placed left-justified at the rightmost position within the level, minimizing the residual horizontal space. The main idea behind BFDH is to minimize the wasted space in each level by evaluating the available space towards the right end of the level, where the rectangle will be packed. For each level, the algorithm calculates how much unused horizontal space would remain if the rectangle were placed on that level and then chooses the level that leaves the smallest leftover space. This results in a packing arrangement where the remaining space in each level is more evenly utilized, compared to the FFDH approach, which might leave larger gaps in some levels due to its more straightforward search for available space. By minimizing the leftover horizontal space, BFDH often achieves a more compact packing solution with less wasted area.

textbfFlowchart for BFDH:


Start
Sort rectangles by height (non-increasing)
For each rectangle: Search all levels to find the best fit
Calculate horizontal space left if rectangle is placed in each level
Find level with minimum horizontal space left
Can rectangle fit in any existing level?
Place rectangle on new level
Repeat for next rectangle
End

Advantages:

- BFDH generally uses the available space more efficiently than FFDH by minimizing the leftover horizontal space on each level, leading to a more compact arrangement and potentially less waste.

- By placing items on the level with the smallest residual space, BFDH can reduce gaps between rectangles, resulting in a higher packing density and reduced material waste.

- The algorithm searches across all levels, allowing it to adapt better to different packing configurations and avoid the inefficiencies of FFDH, where once a rectangle is placed, it can't be moved.

- BFDH performs well in cases where the shape of the rectangles leads to small, fragmented gaps that are difficult to fill. By searching for the best level, it can avoid such fragmentation.

Disadvantages:

- The main drawback of BFDH is its increased computational complexity. Since the algorithm must search through all existing levels for the best fit (instead of just the first available

spot), it requires more calculations and may take longer, particularly for larger problems.
- BFDH requires more memory to store the current packing state, as it needs to keep track of the residual spaces in each level to determine the best place to fit the next rectangle.
- The algorithm's need to scan all levels for the optimal placement results in slower execution compared to FFDH
- For very large instances, the increased computation time can make BFDH less practical compared to simpler methods or even more sophisticated approaches.

Comparison between FFDH and BFDH:

Aspect	**FFDH**	**BFDH**
Approach	Greedy, first available fit	Greedy, best available fit based on residual space
Sorting	Items sorted by height in non-increasing order	Items sorted by height in non-increasing order
Space Utilization	Can leave larger gaps and waste	More efficient in minimizing waste and space
Execution Speed	Faster, as it only searches for the first available space	Slower, due to searching all levels for best fit
Implementation Complexity	Simple and easy to implement	More complex, requires tracking residual space
Solution Quality	Suboptimal, but can be close to optimal in many cases	Better packing density, less waste
Memory Usage	Low, only requires storing the current packing state	Higher, due to tracking residual space on each level
Suitability for Large Problems	Performs well for medium-sized problems	More suitable for smaller problems due to higher computation time
Waste Generation	May result in higher waste due to inefficient packing	Minimizes waste more effectively by fitting items into the smallest residual space
Best Use Case	When a fast, approximate solution is needed for medium-sized problems	When packing efficiency is critical, and the problem size is manageable

4 Modeling

Material usage efficiency is a critical factor in industries that rely on raw materials, as waste directly impacts profitability and environmental sustainability.

4.1 Problem definition

Given a set of stock sheets of fixed dimensions (width W, height H) and a list of smaller rectangular items with specified dimensions (w_i, h_i) and required quantities q_i, the goal is to determine how to cut the stock sheets to satisfy all requirements while minimizing waste (unused space on the stock sheets) or the number of stock sheets used.

4.2 Componets of the model

Input:

- W, H : Width and height of stock sheets.
- w_i, h_i : Width and height of item i
- q_i : Quantiy of item i

Decision Variables:

- x_i, y_i : Coordinates specifying the bottom-left position of item i on the stock sheet.
- r_i : Binary variable indicating whether item i is rotated (1 if rotated, 0 otherwise).
- u_j : Binary variable, 1 if stock sheet j is used, 0 otherwise.

Objective Function:

Minimize waste or the number of stock sheets : $\sum u_j$

Constraints:

- Fit Constraints: Ensure all items fit within the dimensions of the stock sheet:

$$x_i + w_i \leq W, \quad y_i + h_i \leq H, \quad \forall i$$

- Non-Overlap Constraints: Ensure no two items overlap

$$(x_i + w_i \leq x_k) \vee (x_k + w_k \leq x_i) \vee (y_i + h_i \leq y_k) \vee (y_k + h_k \leq y_i), \quad \forall i \neq k$$

- Demand Constraints: Ensure the required quantities of all items are met

$$\sum_j q_i \geq \text{Required quantity of each item } i$$

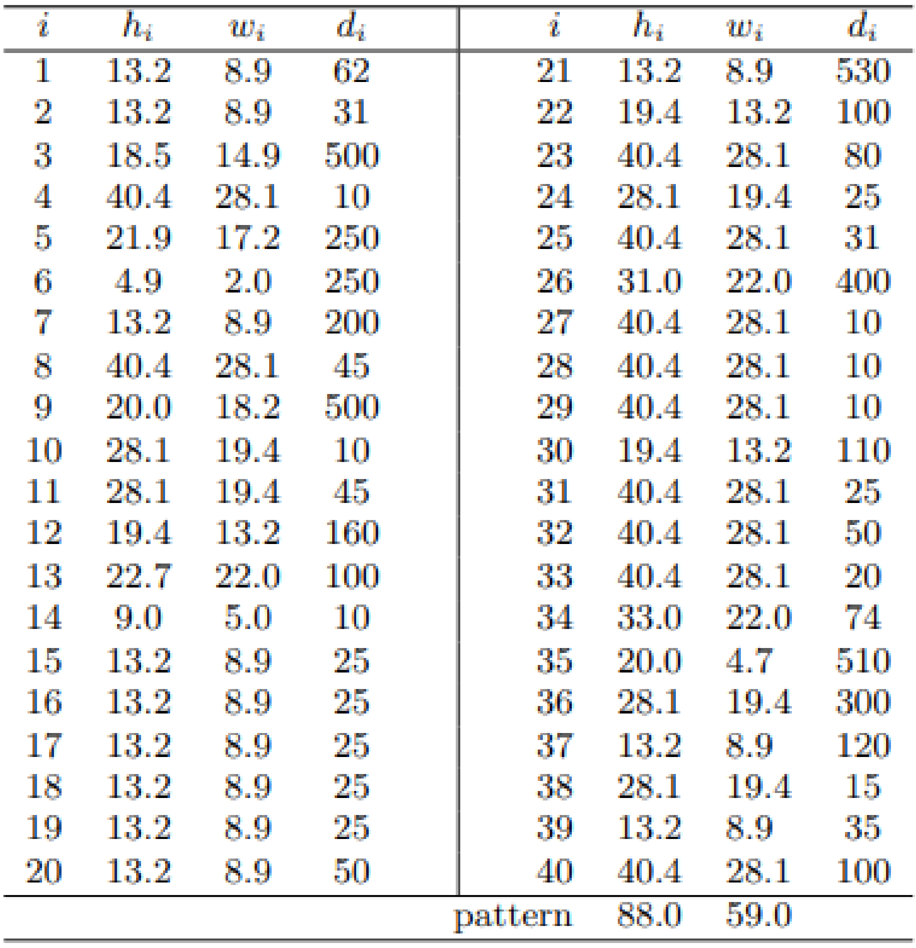
- Binary Constraints: Define whether stock sheets are used

$$u_j = 1 \text{ if any items are placed on sheet } j, \quad u_j \in \{0, 1\}$$

5 Case study

The objective of this work was to deal with a real-world application. Every day, the company receives a dataset composed of 40 to 50 images. The size of the patterns is always equal to $88 \times 59\,\text{cm}^2$. The spacing between two side-by-side images must be equal to 1.6 cm. Thus, we added 0.8 cm to the height and width of each image. The costs are defined as follows:

- $C_{su} = 20\$$
- $C_{ss} = 1\$$



i	h_i	w_i	d_i	i	h_i	w_i	d_i
1	13.2	8.9	62	21	13.2	8.9	530
2	13.2	8.9	31	22	19.4	13.2	100
3	18.5	14.9	500	23	40.4	28.1	80
4	40.4	28.1	10	24	28.1	19.4	25
5	21.9	17.2	250	25	40.4	28.1	31
6	4.9	2.0	250	26	31.0	22.0	400
7	13.2	8.9	200	27	40.4	28.1	10
8	40.4	28.1	45	28	40.4	28.1	10
9	20.0	18.2	500	29	40.4	28.1	10
10	28.1	19.4	10	30	19.4	13.2	110
11	28.1	19.4	45	31	40.4	28.1	25
12	19.4	13.2	160	32	40.4	28.1	50
13	22.7	22.0	100	33	40.4	28.1	20
14	9.0	5.0	10	34	33.0	22.0	74
15	13.2	8.9	25	35	20.0	4.7	510
16	13.2	8.9	25	36	28.1	19.4	300
17	13.2	8.9	25	37	13.2	8.9	120
18	13.2	8.9	25	38	28.1	19.4	15
19	13.2	8.9	25	39	13.2	8.9	35
20	13.2	8.9	50	40	40.4	28.1	100
				pattern	88.0	59.0	

Figure 1: Daily dataset of the company.

5.1 Results of 2 models (FFDH and BFDH)

The results after running Model 1 (FFDH) to address the above Case Study are as follows:

```
    Patterns: 14
    Sheets Used: 40
    Total Time (seconds): 70
    Sum of Remainings: 4056.459999999999
    Total Cost ($): 4336.459999999999

```

Program 1: Result of model FFDH

The results after running Model 2 (BFDH) to address the above Case Study are as follows:

```
    Patterns: 14
    Sheets Used: 40
    Total Time (seconds): 70
    Sum of Remainings: 3142.6499999999996
    Total Cost ($): 3422.6499999999996

```

Program 2: Result of model BFDH

5.2 Comparison between 2 models

Both methods produced solutions with an identical number of patterns (14) and sheets used (40), reflecting the same level of efficiency in pattern distribution and sheet utilization. However, the FFDH method demonstrated a notable advantage in terms of resource optimization and cost efficiency.

The total remaining unused space for FFDH was 3142.65 cm^2, significantly lower than the 4056.46 cm^2 observed for BFDH. This difference indicates that FFDH made more effective use of the available sheet area, reducing waste. Consequently, the total cost for FFDH amounted to \$3422.65, which is approximately 20% lower than the \$4336.46 incurred using BFDH, due to the reduced cost associated with unused space.

Both methods required the same processing time, with a total of 70 seconds for generating the patterns, assuming an average of 5 seconds per pattern. This indicates that the computational efficiency of the two heuristics is comparable. However, the higher efficiency of FFDH in minimizing waste and overall costs makes it the superior choice for this problem. These results suggest that for datasets of similar characteristics, FFDH should be preferred to optimize material usage and reduce operational expenses, while maintaining a competitive runtime for decision-making processes.

5.3 Evaluation of performance

FFDH Performance :

FFDH excels in its systematic approach by prioritizing the placement of larger items earlier, thereby achieving better compaction and reducing waste. In this analysis, FFDH minimized the remaining unused space to $3142.65\,\mathrm{cm}^2$, which is approximately 22.5% less than the unused space left by BFDH.

This efficient space utilization directly translated into cost savings, with FFDH reducing the total cost to $3422.65, compared to $4336.46 for BFDH. The reduction in waste and costs demonstrates that FFDH is particularly well-suited for datasets requiring tighter optimization, making it a highly efficient and cost-effective solution for the company's daily operations.

BFDH Performance :

On the other hand, BFDH, which attempts to fit items into the tightest available spaces, was less effective with this dataset. While it matched FFDH in the number of patterns (14) and sheets used (40), the remaining unused space was significantly higher at $4056.46\,\mathrm{cm}^2$.

This resulted in increased costs due to higher waste, with the total cost reaching $4336.46. While BFDH's strategy can be advantageous for datasets with more uniform dimensions or less variation in item sizes, it underperformed in this case, where item dimensions varied widely.

5.4 Recommendations for Enhancements and Further Development

Incorporate Rotational Fit Analysis Both FFDH and BFDH currently assume fixed orientations for items (width and height). Incorporating a rotation heuristic that allows items to be rotated by 90° can significantly improve space utilization. For instance:
- If an item fits better when rotated, the algorithm should dynamically adjust its orientation.
- This enhancement would be particularly useful for datasets with irregular or diverse dimensions.

Hybrid Heuristic Approaches

A hybrid approach that combines the strengths of FFDH and BFDH can improve performance:
- Use FFDH for an initial pass to prioritize larger items and minimize wasted space.
- Apply BFDH on the remaining items to efficiently utilize tighter spaces.

This combination would leverage the compaction ability of FFDH with the precise fitting capabilities of BFDH.

Dynamic Item Reordering

Current implementations rely on static sorting of items based on dimensions. Introducing dynamic reordering during the process could enhance performance:
- Periodically reassess and reorder items based on remaining available space in the pattern.
- Dynamically switch to different heuristics (e.g., BFDH after a few iterations of FFDH) to handle changing space constraints

Layered Space Optimization

Introduce a layered optimization strategy to handle multi-dimensional packing:
- Divide the pattern into sub-regions (layers) and allocate items based on their fit within each region.
- Optimize each layer individually while ensuring the overall utilization is maximized.

This method can be particularly beneficial for complex datasets with a mix of small and large items.

Genetic Algorithm or Metaheuristic Integration

Augmenting the heuristics with metaheuristic techniques such as genetic algorithms, simulated annealing, or particle swarm optimization could yield better solutions:
- Generate multiple initial solutions using FFDH or BFDH and iteratively refine them through a metaheuristic.
- This approach could explore a wider solution space and escape local minima.

Adaptive Cost Function

Introduce a dynamic cost function that prioritizes both space utilization and cost minimization:

- Penalize patterns with higher unused space more heavily.

- Adjust weightage between sheet cost (C_{su}) and waste cost (C_{ss}) based on real-world operational priorities.

Parallel Processing

Implement parallelized versions of FFDH and BFDH to handle larger datasets more efficiently:

- Split the dataset into smaller batches and process them simultaneously.

- Merge results while ensuring global optimization is maintained.

Machine Learning Assistance

Use machine learning models to predict the best-fitting heuristic for each item based on historical datasets:

- Train a model to classify items as ”best fit for FFDH” or ”best fit for BFDH.”

- Dynamically switch between the heuristics based on the model’s predictions during the process.

6 Conclusion

The 2D-CSP has become increasingly relevant across diverse industries like steel, textiles, and aluminum, where efficient resource utilization directly impacts profitability and environmental sustainability.

The study explored both algorithmic and heuristic methods to tackle the 2D-CSP. Classical approaches like Linear Programming and column generation have provided foundational tools for solving these problems, while modern techniques such as branch-and-cut and nested decomposition have extended their applicability to more complex cases. Simultaneously, heuristic methods, including First-Fit Decreasing Height (FFDH) and Best-Fit Decreasing Height (BFDH), offer practical and computationally efficient alternatives for generating near-optimal solutions. These methods, though suboptimal, are particularly effective for large-scale problems where exact methods may be computationally prohibitive.

A detailed analysis of FFDH and BFDH revealed their respective strengths and weaknesses. FFDH excels in simplicity, speed, and ease of implementation, making it suitable for medium-sized problems where a fast and approximate solution is sufficient. However, its tendency to leave larger gaps and generate higher waste makes it less efficient in material utilization. In contrast, BFDH achieves better packing density by searching for the best available fit, minimizing residual space and waste. While this results in a higher-quality solution, the increased computational complexity and memory usage can make BFDH less practical for very large datasets. A comparative analysis highlights that FFDH is better suited for scenarios prioritizing speed, while BFDH is more effective when packing efficiency is critical, and computational resources are sufficient.

In modeling the 2D-CSP, an optimization framework was defined, integrating decision variables and constraints to ensure items fit within stock sheets, avoid overlap, and meet demand requirements. By incorporating advanced techniques such as rotation and adaptive layout strategies, the model balances practical implementation with theoretical rigor. The flexibility of this framework allows it to accommodate diverse problem scenarios, supporting industries in their pursuit of cost reduction and sustainable operations.

In conclusion, the 2D-CSP remains a critical problem in operations research, with widespread applicability and significant economic and environmental implications. The combined use of exact and heuristic methods offers a powerful toolkit for addressing the challenges posed by this NP-hard problem. Future advancements, such as hybrid optimization frameworks and machine learning-based strategies, promise to further enhance the efficiency and adaptability of solutions. By leveraging these innovations, industries can optimize material usage, reduce waste, and improve overall operational efficiency, contributing to both profitability and sustainability goals.

7 References

1. A survey and comparison of guillotine heuristics for the 2D oriented offline strip packing problem.
2. Approaches to real world two-dimensional cutting problems.
3. A Genetic Algorithm to Solve a Real 2-D Cutting Stock Problem with Setup Cost in the Paper Industry.
4. J. Jylanki. A thousand ways to pack the bin - a practical approach to two-dimensional rectangle bin packing. research report, 2010..
5. J. Kallrath, S. Rebennack, J. Kallrath, and R. Kusche. Solving real-world cutting stock-problems in the paper industry: Mathematical approaches, experience and challenges. European Journal of Operational Research, 238(1):374–389, 2014. ..